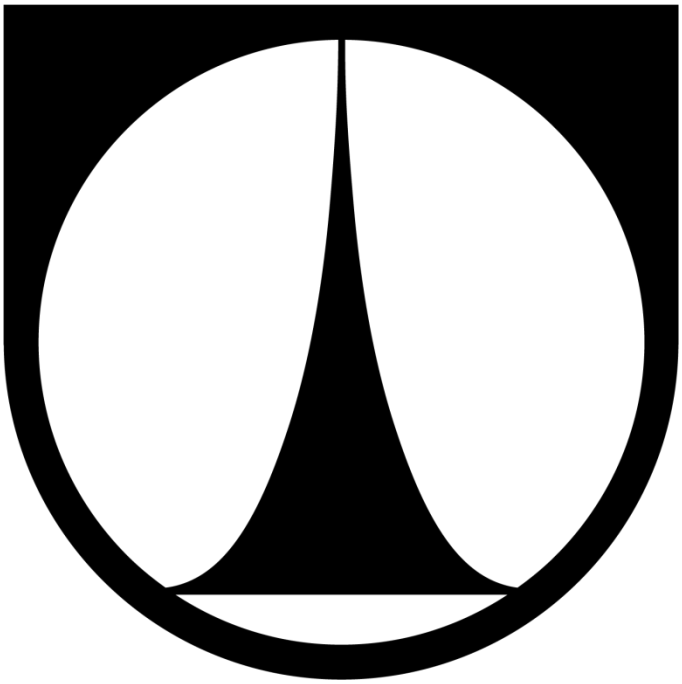
TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta



DIPLOMOVÁ PRÁCE

2015 **Bc. Petr Křivský**

TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

Studijní program: N6209 - Systémové inženýrství a informatika

Studijní obor: 6209T021 - Manažerská informatika

Optimalizace kontroly vývoje SW v systémech SAP

Control of SW Development and Optimization for SAP Systems

Bc. Petr Křivský

Vedoucí práce: doc. Ing. Klára Antlová, Ph.D.

Konzultant: Ing. Tomáš Slabý

Počet stran: 67 Počet příloh: 2

Datum odevzdání: 07. 05. 2015

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Děkuji vedoucí diplomové práce paní doc. Ing. Kláře Antlové, Ph.D. za ochotu, rady a možnost pod jejím vedením uskutečnit tuto práci.

Především děkuji Ing. Tomáši Slabému a celému oddělení EOA firmy ŠKODA AUTO a.s. za přijetí do kolektivu při vykonávání praxe a vytvoření vhodného prostředí pro vypracování diplomové práce.

Anotace

Informační systém je nutným předpokladem k úspěšnému fungování firem ve všech oblastech podnikání. Každý podnik je ve své podstatě unikátní a informační systém je nutností navrhnout na míru jednotlivým požadavkům podniků. Systém je nutné konfigurovat a neustále upravovat. Konfigurační práce jsou součástí práce se systémem a informační systém obsahuje k tomu určené nástroje. Tyto konfigurační a vývojové práce je potřeba kontrolovat za účelem udržení kvality celého systému. Tato diplomová práce se zabývá optimalizací procesu kontroly vývoje software v informačních systémech SAP.

Klíčová slova: SAP, ERP, SAP Solution Manager, Zákaznický vývoj software, Kontrola kvality vývoje

Anotation

The information system is a necessary prerequisite for successful functioning of companies in all business areas. Every company is unique and information system has to be designed for specific company needs. Information system is required to be configured and edited. The configuration is a part of the working with the system and system includes dedicated tools to do that. The configuration and development must be controlled to maintain the quality of the whole system. This thesis deals with the control process of software development and optimization for SAP systems.

Keywords: SAP, ERP, SAP Solution Manager, Custom code development, Quality control

Obsah

Seznam obrázků

Seznam zkratek a termínů

CRM - Customer relationship management

CCM - Custom Code Management

ERP - Enterprise Resource Planning

ISO - International Organization for Standardization

IT - Informační technologie

KMS - Knowledge Management System

KPI - Key performance indicator

QA - Quality Assurance

SCM - Supply Chain Management

SOLMAN - Aplikace SAP Solution Manager – nástroj pro podporu správy a rozvoje aplikací SAP

SW - Software

TCO - Total Cost of Ownership

Úvod

Informační systém v podnikové sféře tvoří základní nástroj pro fungování podnikových procesů. Kvalita informačního systému a rozsah jeho využívání do značné míry rozhoduje o úspěšnosti podniku. Na trhu existuje množství celopodnikových ERP systémů a zákazník (v podobě podniku) nemusí vyvíjet systém od začátku. Tyto celopodnikové systémy jsou založeny na principu generalizace. Jednotlivé požadavky podniků jsou zobecněny, aby je bylo možné použít v co nejvíce případech. Zákazníci dotvářejí detaily požadavků procesním nastavením systému dle vlastních potřeb.

Informační systém se musí stále přizpůsobovat potřebám a procesům firmy, ta musí reagovat na situaci na trhu a sama přicházet s inovacemi. Pokud firma nepřichází s vlastní iniciativou, ztrácí konkurenční výhodu. Velké softwarové firmy, které dodávají univerzální řešení pro široké spektrum zákazníků, nedokáží dodávat řešení přímo na míru jednotlivým podnikům.

Změny a úpravy informačního systému nejsou výjimka, jsou nevyhnutelné. Systém je nutné neustále upravovat a zvyšovat aplikační hodnotu. Kvalita informačního systému závisí na kvalitě jednotlivých softwarových částí, jež tvoří součást celého systému. K úpravám stávajícího řešení na základě potřeb, k tvorbě nového řešení a rozvoji v podobě aplikací aj. slouží zákaznický vývoj. Pro zajištění kvality zákaznického vývoje je nutné provádět audit zákaznického kódu. Tato práce se zabývá kontrolou kvality vývoje v informačních systémech SAP. Snaží se přiblížit problematiku životního cyklu vývoje informačního systému a jednotlivých aplikací. Popisuje metodiky využívané k tvorbě informačního systému.

Práce se zaměřuje na možnosti kontroly vývoje a její optimalizaci. Zvláště uvádí metody a nástroje využívané při kontrole zákaznického vývoje v systémech SAP založených na programovacím jazyce ABAP. Popisuje řešení kontroly zákaznického vývoje uvnitř výrobního podniku.

1 Vývoj software a informačních systémů

1.1 Informační systém

Systém je v obecně přijaté definici chápán jako množina prvků a vazeb. Vazby mezi prvky představují jednosměrné nebo obousměrné spojení mezi nimi. Systém charakterizují vstupní a výstupní vazby s okolím, kterými informace získává a jiné informace do okolí předává. Informacemi rozumíme data, která slouží pro rozhodování a řízení. Data chápeme jako signály, které vypovídají o situacích a stavech sledovaných objektů. Informace jsou tedy interpretovaná data, která jsou použita pro další rozhodování. Stejná data, která jsou různě interpretovaná, mohou představovat pro různé uživatelé různé informace. [1]

Informační systém lze chápat jako systém vzájemně propojených informací a procesů, které s těmito informacemi pracují. [2]

Okolí informačního systému tvoří veškeré objekty, které změnou svých vlastností ovlivňují samotný systém, a také objekty, které naopak mění své vlastnosti v závislosti na systému. V podnikové praxi si můžeme představit informační systém jako kombinaci hardwaru, softwaru, infrastruktury a trénovaného personálu, který systém využívá na plánování, kontrolu, koordinaci a rozhodování v organizaci.

Informační systém (IS) definujeme jako uspořádání vztahů mezi lidmi, datovými a informačními zdroji a procedurami jejich zpracování za účelem dosažení stanovených cílů. [1]

V současné praxi je informační systém nutným předpokladem k úspěšnému fungování firem ve všech oblastech podnikání. Podniky jsou závislé na kvalitních a včasných informacích. V posledních letech se až několikanásobně zvyšují objemy dat, které společnost produkuje. Význam informace pro podnik se stává klíčový a jejich včasné zpracování určuje efektivnost a konkurenceschopnost firmy.

Tato situace je způsobena především prudkým růstem informatizace společnosti a právě proto se v posledních letech výrazně, a to až několikanásobně, zvyšují objemy finančních prostředků investovaných do inovace informačních systémů a informační technologie (IS/IT).

1.2 Business proces

Pro lepší pochopení fungovaní dnešních informačních systémů v podnikové praxi je nutné definovat pojem business proces. Proces je definován dle normy ČSN EN ISO 9000:2006 jako „*soubor vzájemně souvisejících nebo vzájemně působících činností, který přeměňuje vstupy na výstupy*.“ [3]

Pojem proces je v dnešní době velmi používané slovo a setkáme se s ním téměř všude. Je používáno především manažery a to nejvíce v organizacích, které se zabývají informačními technologiemi. Slovo proces se stalo tak běžným, že se zapomíná na jeho základní význam. Procesní myšlení se liší od tradičního vnímání života organizace. Nutností je například opustit hierarchické organizační struktury. U procesů je základním kritériem čas. Pokud mluvíme o posloupnosti činností, máme na mysli časový úsek, který je rozdělen do jednotlivých úseků. Popis procesu je popisem procesním. Popisován je postup, nikoli věc. Základem pro procesní řízení je pochopení základní logiky businessu. Je nutné pochopit posloupnost činností a jejich souvislostí ve vazbě na strategické hodnoty firmy. Organizační struktura a informační systém pak slouží pouze jako infrastruktura pro podporu business procesů.

Dynamika ve fungování a řízení organizace se stala klíčová. V praxi je nutností rychle reagovat na vývoj nových technologií. Inovace s příchodem nových technologií je možné rozdělit na dvě roviny. První rovina umožňuje optimalizovat nebo zvýšit výkon v jednotlivých prvcích výkonu. Ve druhé rovině můžeme měnit pořadí částí pracovních postupů za cílem optimalizace. [4]

Ve firmách s implementovaným systém kvality dle ISO 9000 nebo ISO 9001 je tento systém kvality implementován na základě procesů. Zlepšování procesů je pak základním principem fungování tohoto systému. Procesní řízení je soubor činností týkajících se plánování a sledování výkonnosti především realizačních firemních procesů. Priorita procesního řízení je proces samotný. Při stanovení procesu se pak neklade důraz na organizační struktury. Nejprve je definována sekvence činností, které tvoří proces a až následně se stanoví, kdo jednotlivé činnosti provádí a jaká je organizace těchto pracovníků. Výsledkem je vnitřní struktura podniku, která je co nejvíce přizpůsobena podpoře podnikových procesů.

Tato práce se zabývá optimalizováním procesu kontroly kvality software. Pro tento proces, jakožto pro ostatní metodiky řízení zlepšování kvality, platí obecný postup.

a) **definice problému** – V této fázi dochází ke sběru a porozumění potřebám zákazníka. Je potřeba identifikovat klíčové požadavky. Na požadavcích zákazníka je postaven celý tento systém.
b) **provedení měření** – v tomto kroku je třeba identifikovat vstupy, výstupy a proměnné, které nejvíce ovlivňují celý proces. Výstupy tohoto procesu jsou metriky. Mezi hlavní metriky patří například čas, náklady a kvalita.
c) **analýza** – za pomocí metrik a analýzy procesu se zjišťují příčiny problému. K tomu slouží metody jako např. brainstorming, diagram příčin a následků (Ishikawa diagram), stromový diagram atd.
d) **zlepšování** – tato fáze se zabývá vymýšlením a hodnocením jednotlivých variant. Následuje implementace řešení.
e) **řízení** – v této fázi se sbírají data o implementovaném řešení a pomocí standardizací procesu je zajištěna dlouhodobá podpora řešení. Případně se navrhují další kroky. [5]

Procesní řízení nahrazuje zastaralý systém funkčního řízení, který má řadu nevýhod. Ve firmách zastávajících funkční řízení často nefunguje komunikace napříč firmou. Funkční řízení nepodporuje týmovou spolupráci a dává do popředí potřeby jednotlivých útvarů nad celkovým fungování a úspěchem firmy. Řízení firmy je nepružné a byrokratické. Informace jsou často používány pouze v rámci jednoho oddělení a mohou sloužit ke konkurenčnímu boji částí funkčního systému. V neposlední řadě existuje tendence vytvářet nadbytečná pracovní místa a funkce. Procesní řízení tyto problémy řeší a přináší další výhody.

1.3 Typy informačních systémů

Informační systémy v podnikové praxi se dále dělí podle účelu použití. Různé informační systémy sdružují různé skupiny business procesů.

CRM (Customer Relationship Management) – CRM je systém na podporu vztahů se současnými a budoucími zákazníky. Zahrnuje technologie pro správu a automatizaci prodejů, marketingu, servisu a technické podpory.

ERP (Enterprise Resource Planning) - ERP je informační systém na správu, integraci a automatizaci business procesů podniku. Jedná se o většinu běžných činností podniku z oblastí prodeje, nákupu, logistiky, lidských zdrojů, správy majetku, účetnictví a plánování výroby. ERP poskytuje celkový pohled na základní činnosti podniku obvykle v reálném čase s pomocí užití databází spravovaných databázovými systémy. Tento podnikový informační systém monitoruje zdroje: materiál, peníze, kapacitu výroby s cílem zajištění potřeb trhu vlastního podniku.

SCM (Supply Chain Management) – Každý výrobek, který dorazí k zákazníkovi, musí projít cestou přes řadu jednotlivých organizací. Jedná se o výrobce, dodavatele, zákazníky apod. Tento systém s pomocí integrace všech článků umožňuje efektivní správu celého dodavatelského řetězce. Cílem tohoto systému je maximalizovat hodnotu pro zákazníky a dosažení udržitelné konkurenční výhody. Všechny články řetězu spolu vědomě spolupracují a snaží se najít a udržovat co nejefektivnější řešení. Organizace, které jsou spojeny fyzickými toky materiály, jsou také spojeny informačními toky. Informační toky jsou velmi důležité a umožňují všem partnerům v dodavatelském řetězci, aby koordinovaly své dlouhodobé plány a řídili tok zboží a materiálu na denní bázi. [6]

KMS (Knowledge Management System) – pomáhá organizaci ve sběru, zaznamenávání, uspořádávání, získávaní a šíření znalostí. Řadí se sem např. postupy, praxe a dovednosti. Znalosti jsou začleněny do organizace a pomáhají při různých činnostech a představují cennou komoditu. S jejich pomocí je možné snížit náklady na čas při řešení podobných problémů nebo na zaškolení nového pracovníka. [7]

1.4 Software

Software je programové vybavení počítače. Skládá se z instrukcí, které je schopen počítač přečíst. Software je opak fyzického vybavení počítače - hardware. Software nemůže být bez hardware používán.

Na nejnižším stupni se zpracovávaný kód skládá ze strojového kódu, který se liší v závislosti na architektuře jednotlivých procesorů. Strojový kód obsahuje soubor binárních hodnot tvořících jednotlivé instrukce. Tyto instrukce řídí procesor, který je zpracovává a provádí požadované operace. Software není běžně psán ve strojovém kódu, nýbrž v přívětivějším stavu. K tomuto účelu slouží programovací jazyky, které jsou překládány pomocí překladačů do strojového kódu.

Software můžeme rozdělit do několika kategorií podle různých kritérií. Např. účel užití:

a) **Aplikační software** – umožňuje počítači vykonávat speciální úkoly nad rámec běžných operací počítače.
b) **Systémový software** – systémový software je programové vybavení počítače mezi aplikačním softwarem a hardwarem počítače. Umožnuje s počítačem pracovat, spouštět aplikace atd.
c) **Škodlivý software** – software vyvinutý za účelem poškození nebo vniknutí do počítačového systému. Tento software je nežádoucí. V dnešní době je často tvořen za účelem získání osobních dat. [8]

1.5 Vývoj software

Za účelem lepšího pochopení problematiky kontroly vývoje software je nejprve nutné představit samotný postup vývoj software a různé metodiky používané v praxi. Vývoj software představuje velmi problematickou a náročnou činnost. Od počátku využití počítačů koncem 40. let dvacátého století prošel software mnoha stádii vývoje. Hlavním faktorem, který historicky ovlivnil vývoj software, bylo představování stále nových a výkonnějších počítačů. Vývoj hardware šel ruku v ruce s vývojem software. Zmenšování a zrychlování počítačů od velkých halových mainframu k malým, rychlým a spolehlivým strojům přestavovalo stále větší nároky na vývoj software. V 50. letech 20. století se poprvé objevilo spojení softwarové inženýrství. Jako obor vzniká softwarové inženýrství v 70. letech 20. století.

Softwarové inženýrství je souhrn zkušeností a inovací, které slouží pro psaní kvalitních programů při optimálním využití zdrojů. Tyto znalosti tvoří základní principy

softwarového inženýrství. Softwarové inženýrství je inženýrská disciplína zabývající se praktickými problémy vývoje rozsáhlých softwarových systémů. [9]

V počátcích vývoje programů byl používán model „Napiš a oprav“ volně přeloženo z anglického spojení „Code and fix“. Tento model je zcela základní a název dostal až později. Smysl tohoto modelu je produkování kódu bez většího přemýšlení. V určité fázi přešel program do testování a pracovalo se na odstranění chyb před dokončením produktu. Tento model nebylo možné dlouhodobě udržet. Například zcela postrádá sběr požadavků od zákazníka a návrhu software před samotným programováním.

Do tohoto období vznikaly programy šité na míru daného počítače a architektury, většinou pevně zapsané do paměti počítače. S dostupností hardware se začíná rozvíjet i software. Začínají se vytvářet uživatelsky příjemnější programy. Knihovny a další částí programů se začínají prodávat jako doplňky. V tomto období se také setkáváme s tzv. softwarovou krizí, která trvá až do 80. let 20. století. Tato doba narazila na problémy v dosavadním přístupu ve vývoji software. Projekty přesahovaly rozpočet nebo požadovanou dobu vývoje a programy nedosahovaly požadovaných kvalit. Mezi problémy patřila nízká kvalita programátorů, nefungující komunikace mezi vývojáři a zákazníkem, nemožnost inovací a celková neefektivita vývoje. Mnoho projektů, které byly po problémech dodány, se nikdy nepoužily. Proto se začalo pracovat na nápravě. V roce 1968 na konferenci NATO se začal utvářet nový směr vývoje a byl zaveden pojem softwarové inženýrství. Jako řešení těchto problémů se začaly používat techniky a také softwarově-inženýrské metodiky na vývoj software, jako například specifikace, návrh, testování, nebo modely životního cyklu software.

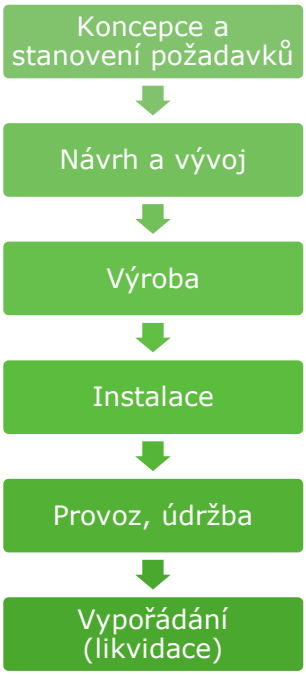
1.6 Životní cyklus vývoje softwaru

Software si představujeme jako výrobek, ale tato představa není úplně přesná. Používáním i způsobem výroby se odlišuje od klasických produktů. Klasický produkt můžeme definovat řadou přesně měřitelných parametrů. U automobilu to jsou např.: výkon motoru, spotřeba, výbava. U softwaru toto měření není jednoznačné. Velikost, kterou software na disku zabírá, není určujícím faktorem jeho kvality a zákazník nemůže porovnat jeho kvality s ostatními. Definice kvality software pomocí konkrétních parametrů není vždy

jednoduchá. Software je ve své podstatě nehmotný. Proces jeho vývoje se tak odlišuje od tradičních produktů. Běžné postupy používané v průmyslové praxi proto nejsou vhodné pro vývoj softwaru. Software se liší také způsobem využití. Zákazník u software požaduje specifické funkce už při zadání jeho vývoje a během používání může žádat o doplnění dalších nadstandartních funkcí, které nemusí přímo souviset s původním účelem programu.

Aby byly zohledněny specifické požadavky nejen na vývoj, ale také na udržování a rozšiřování software, vznikají modely životního cyklu vývoje softwaru. Tyto modely zohledňují celé období používání softwaru od jeho plánování, realizaci, testování, zavedení a údržbu.

Pojem Životní cyklus produktu jak uvádí norma Management spolehlivosti z roku 2009 [15] je definován jako „*časový interval od stanovení koncepce produktu po jeho vypořádání (likvidaci)*“. Celkem je tvořen šesti etapami.



Obrázek 1. – Etapy životního cyklu
Zdroj: Management spolehlivosti [15] – vlastní zpracování

Etapa koncepce a stanovení požadavků je klíčová pro výsledný produkt. V této etapě, jak její název napovídá, se provádí sběr požadavků na nový produkt. Je uskutečněn

průzkum trhu, definovány technické požadavky a parametry spolehlivosti včetně posouzení jejich spolehlivosti. Vypracuje se finanční analýza projektu a harmonogram prací.

Návrh a vývoj je druhou etapou životního cyklu. Cílem této etapy je návrh produktu z předem zjištěných údajů včetně zpracování podrobné dokumentace. Dokumentace zahrnuje všechny nutné informace k zajištění bezporuchovosti výrobku, kontaktu s vnějšími vlivy, udržovaní atp. Na základě dokumentace je zahájen samotný vývoj.

Etapa výroby vychází z dokumentace vytvořené v předchozí etapě Návrh a vývoj. Na jejím základě je vytvořen produkt. Provádí se funkční kontrola, testování a opravy.

Etapa instalace pokrývá samotnou instalaci produktu v konečném místě provozu. Je zahájen zkušební provoz, který se liší od povahy produktu. Před přechodem do další fáze jsou zajištěny funkce pro možnost údržby. Zákazník je zaškolen a seznámen s vlastnostmi produktu.

Etapa Provoz a servis. Během této etapy je produkt využíván dle domluvených podmínek a specifikace. Jedná se o nejdelší etapu života produktu. Produkt je udržován a nahlášené problémy jsou zaznamenány a odstraněny jejich příčiny. Cílem je udržení kvality a spolehlivosti. V této etapě se můžeme setkat s modernizací. Produkt je modernizován a zdokonalen na základě předchozích zkušenostech s provozem a reakcí s okolím.

Etapa Likvidace. Po ukončení životnosti produktu nastává jeho likvidace. V případě softwaru se jedná o odinstalování a odstranění dalších komponent potřebných k jeho provozu. Je však nutné brát ohled na okolí a nijak jej negativně neovlivnit. [16]

Tato práce se přímo zabývá fází provozu a servisu v rovině informačního systému typu ERP. V nižší rovině zákaznického vývoje se dotýká životního cyklu vývoje softwaru. U zákaznické vývoje, jakožto u celého informačního systému, je nutné se potýkat se všemi etapami životního cyklu od požadavku, plánování, vývoj, instalaci, provoz, údržbu až k vyřazení. Tato práce je zaměřena na kontrolu zákaznického vývoje.

1.7 Metodiky vývoje SW a jejich přístupy

Na počátku existence software stačil na vytvoření produktu i celého jednoduchého systému jediný programátor, který napsal jeho kód. V dnešní době je zapotřebí nejen programátorů, ale také architektů, testovacích techniků, uživatelů a manažerů. Všichni tito lidé musí spolupracovat, aby vytvořili miliony řádků kódu, který tvoří dnešní moderní informační systémy.

Jednotlivé metodiky popisují specifické přístupy jak vyvíjet a udržovat software. Každá metodika detailně určuje jak přistupovat a nakládat se základními životními cykly software. Metodik je velké množství a v této práci jsou uvedeny jen ty nejvýznamnější. Mnohé metodiky jsou zaměřeny jen na některé fáze životního cyklu vývoje software. Většinou se zabývají pouze fází vývoje.

Mezi základní přístupy, kterými se řídí jednotlivé modely, patří:

a) **Sekvenční (lineární) přístup** – je řízen daným plánem, kde jsou jednotlivé kroky prováděny postupně. Patří sem zejména vodopádový model a V-model.
b) **Iterativní přístup** – Tento přístup je založen na rozdělení práce na jednotlivé části (iterace). Zástupcem je například prototypový model, Unified Process (UP), Rapid Application Development (RAD).

Dále se používají kombinace těchto přístupů:

Inkrementální (přírůstkový) přístup – používá se při kombinaci metodik založených na sekvenčním a iterativním přístupu. Cílem je omezení rizik v podobě rozdělení celého projektu na jednotlivé menší části, které se vyvíjí samostatně a přidávají se k již existujícímu celku. Tento přístup přináší výhody v podobě zjednodušení zavádění změn do již existujícího projektu během vývojové fáze.

Spirální přístup - je kombinací sekvenčního a iterativního přístupu. Hlavním zástupcem je Spirálový model.

Agilní přístup – Metodiky agilního přístupu jsou založené na iterativním a inkrementálním přístupu. Jedná se o alternativu k tradičním metodám vývoje software. Agilní přístup umožňuje oprostit se od tradičních metod, které definují jednotlivé procesy

a dávají jasný postup jak vyvíjet software. Agilní programovaní staví do popředí týmovou práci a orientaci na zákazníka. Dávají přednost funkčnosti software před rozsáhlou dokumentací k jednotlivým krokům vývoje a snaží se co nejvíce spolupracovat se zákazníkem, aby bylo možné rychle reagovat na změny. Základ pro metodiky odlehčené od tradičního programování byl sepsán v roce 2001 pod názvem Manifest agilního programování. Hlavním používanou metodikou založenou na agilním přístupu je SCRUM a Extrémní programování.

Principy stojící za Agilním Manifestem:

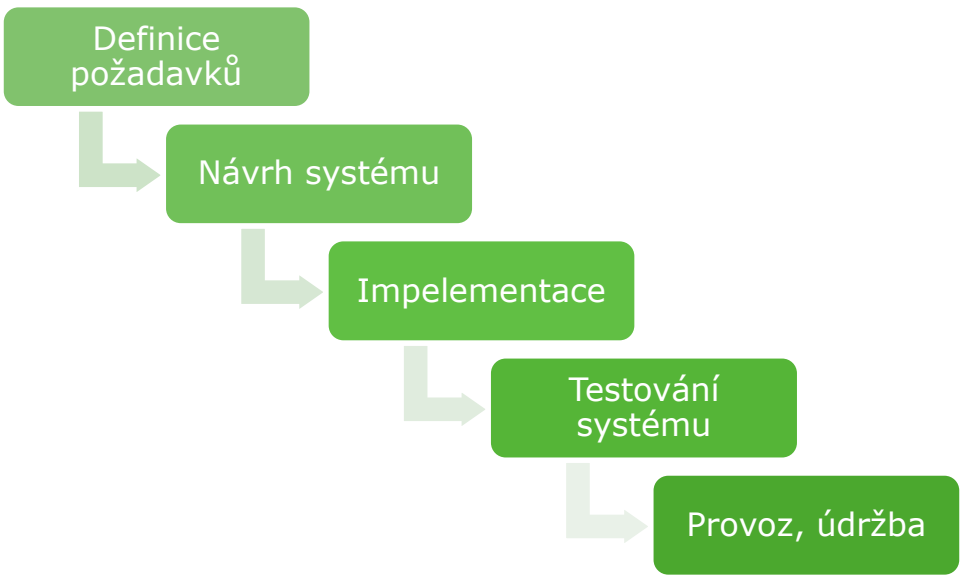
„*Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru. Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka. Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody. Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu. Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci. Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace. Hlavním měřítkem pokroku je fungující software. Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale. Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu. Jednoduchost umění maximalizovat množství nevykonané práce je klíčová. Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů. Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.“* [12]

1.7.1 Vodopádový model

Jak už název modelu napovídá, model připomíná vodopád. Je historicky nejstarší a přichází jako reakce na softwarovou krizi. První formální popis tohoto modelu bylo v roce 1970 Winstonem W. Roycem.

Vodopádový model je založen na lineárně po sobě jdoucích krocích. Cílem jeho zavedení je přivést do vývoje jednotný řád. Jednoznačně definuje jednotlivé vývojové fáze a jejich posloupnost. Přechod mezi fázemi je jednoznačný a projekt se tak nachází pouze v jedné vývojové fázi. Při ukončování jednotlivých fází jsou definované výsledky ověřovány a schváleny na základě předem definovaných akceptačních kritérií uvedených v dokumentaci. Až poté je možné postoupit do další fáze vývoje. Tento model je silně založen na dokumentaci. Dokumenty slouží jako výstup právě ukončené fáze a jsou podkladem pro zahájení fáze následující. Při objevení problému je možné vrátit vývoj do předchozí fáze, tím je ale nutné opakovat všechny následující fáze.

Předností tohoto modelu je jeho jednoduchost a manažerské řízení. K jednotlivým fázím je možné naplánovat a dodržovat přesný časový rámec. Model je možné použít pouze na vývoj, kde jsou dopředu známé všechny požadavky na finální produkt. Zákazník musí jasně definovat své požadavky na začátku vývoje. Pokud jsou problémy odhaleny včas, jejich odstranění není nákladné. Na druhou stranu během vývoje nemá zákazník o podobě software přesnou představu a tudíž má omezenou možnost vznášet připomínky nebo měnit specifikace. Na konci vývoje je software zákazníkovi představen jako finální produkt. Tento model slouží jako teoretický základ ostatním modelům, které jsou pro praktický vývoj software vhodnější. [10]

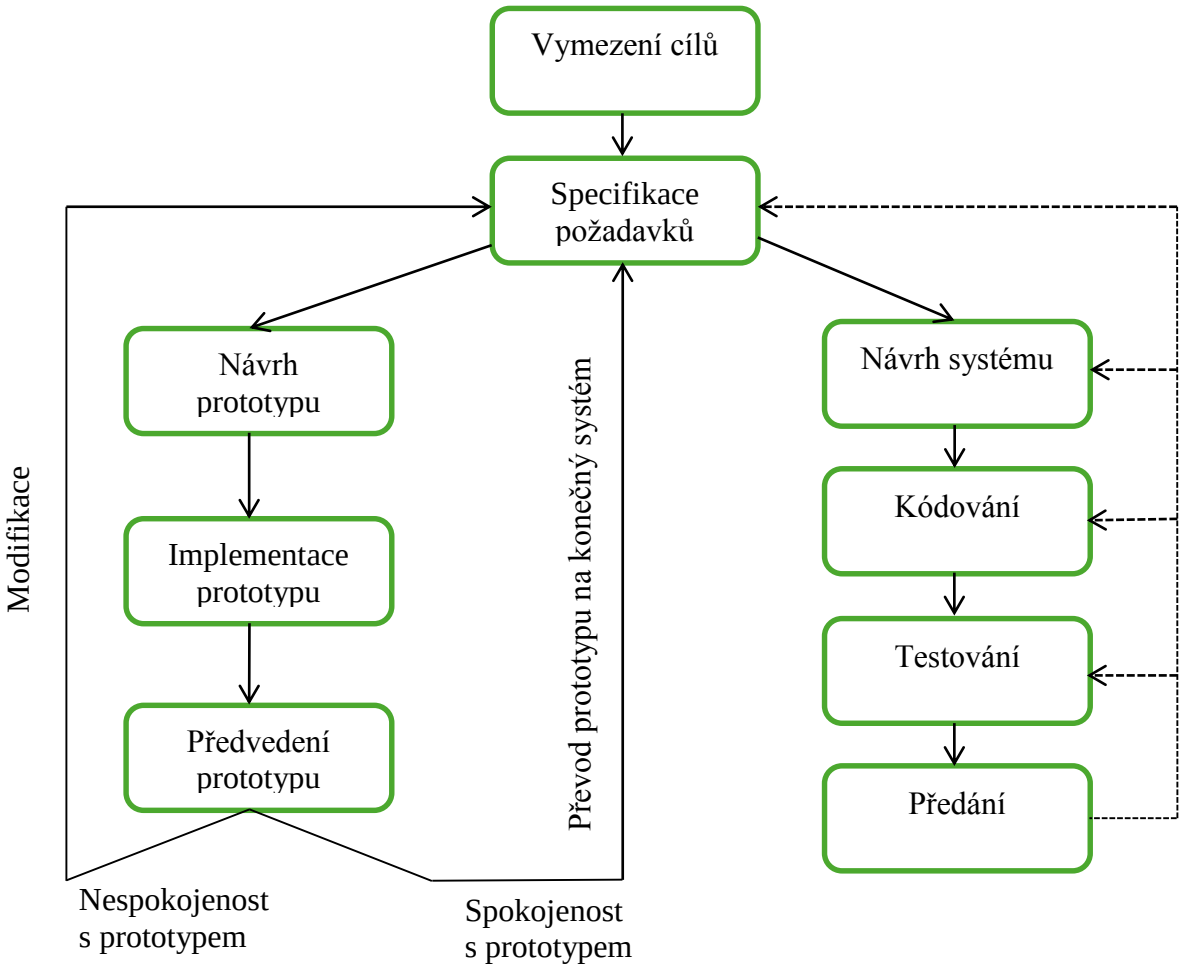


Obrázek 2. – Vodopádový model

Zdroj: Fundamentals of Software Engineering [10] – vlastní zpracování

1.7.2 Prototypový model

Tento model je založen na iterativním přístupu a není používán samostatně. Je součástí větších metodik vývoje jako jejich modifikace. Do vodopádového modelu přináší odstranění jeho základního nedostatku. Tento model předpokládá změnu výchozích požadavků od zákazníka v průběhu vývoje a umožňuje adekvátní reakci. Cílem je v co nejkratším čase představit první prototyp výsledného produktu, ke kterému se může zákazník vyjádřit. Projekt je tímto rozdělen na menší části, tím je usnadněna úprava v průběhu vývoje. Hlavní změnou je urychlení vývoje a co nejrychlejší představení prvního prototypu. Implementace prototypu je v takovém rozsahu funkčnosti, aby byl zákazník schopen reagovat na požadované vlastnosti. Prototyp je modifikován dle upravené specifikace požadavků do doby, než je zákazník spokojen. [11]



Obrázek 3. – Prototypový model

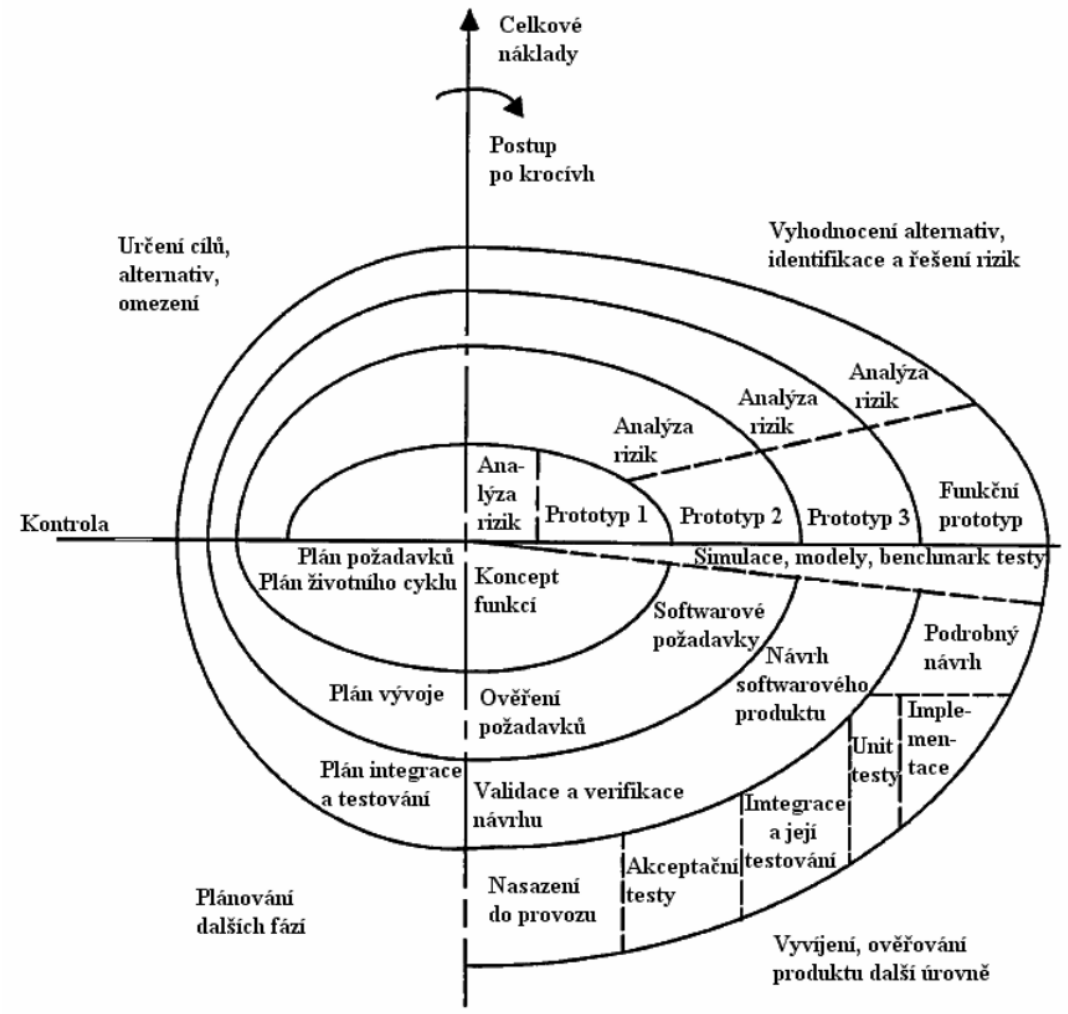
Zdroj: Životní cyklus informačního systému [11] – vlastní zpracování

1.7.3 Spirálový model

Spirálový model byl poprvé definován v roce 1986 v článku (A Spiral Model of Software Development and Enhancement). Jedná se opět o reakci na nedostatky ve spirálovém modelu. Tento model je postaven na řízení riziky. Jednotlivé fáze jsou odděleny důsledným provedením analýzy všech možných rizik, proto je vhodný pro větší projekty. Model je založen na iterativním přístupu, kde se jednotlivé analýzy rizik neustále opakují. Nově vyvinuté části jsou vždy zařazovány k již prověřenému základu produktu. Životní cyklus podle Spirálového modelu rozdělujeme do těchto hlavních částí:

a) určení cílů, alternativ, omezení,
b) vyhodnocení alternativ, identifikace a řešení rizik,
c) vývoj a verifikace další úrovně produktu,
d) plánování následujících fází. [13]

Na konci každé fáze dochází k testování. U velkých projektů je možné využívat automatických testovacích případů a scénářů. Díky včasnému testování je mnoho chyb odhalenu už při rané fázi vývoje.



Obrázek 4. – Spirálový model

Zdroj: http://testovanisoftwaru.cz/wp-content/uploads/2011/07/spirala.png

1.7.4 SCRUM

Název vychází z terminologie hry ragby, kde označuje mnoho hráčů na jednom místě za účelem dopravení míče na danou pozici. Model je založen na agilním přístupu. Definuje flexibilní přístup k vyřešení vývoje, kde tým pracuje společně se stejným společným cílem. Jako agilní přístup je kompatibilní s principy Agilního Manifestu. Jednotlivci a jejich vzájemná interakce je upřednostňována před procesy a nástroji. Větší důraz je kladen na fungující software před úplnou dokumentací, dále komunikace se zákazníkem má přednost před vyjednáváním obchodních podmínek a v neposlední řádě je důležitější včasné reagování na změnu před sledováním daného plánu.

Základní teorií, ze které vychází SCRUM, dle Průvodce Scrumem [21] je „*řízení empirických procesů neboli empirismu. Empirismus tvrdí, že znalost pochází ze zkušenosti a rozhodování založeném na tom, co je známo. SCRUM využívá iterační a inkrementální přístup k optimalizaci předvídatelnosti a řízení rizik. Každá implementace řízení empirického procesu staví na třech pilířích*:“

a) **transparentnost** – všechny důležité aspekty vývoje, musí být přístupné a viditelné všem, kteří mají vliv na výsledek. Tyto aspekty musí používat společný standard.
b) **kontrola** – je nutné provádět kontrolu tak často, aby bylo možné odhalit chyby a odchylky v procesu. Nesmí být však tak častá, aby bránila samotné práci.
c) **adaptace** – pokud je na základě kontroly rozhodnuto, že proces je mimo přijatelné hranice (produkt bude neakceptovatelný), pak je nutné tento proces adaptovat. Změnu je nutné provést co nejdříve.

SCRUM tým se skládá ze tří základních rolí, z vlastníka produktu, vývojového týmu a SCRUM mastera. **Vlastník produktu** je zodpovědný za maximální efektivitu a přidanou hodnotu práce vývojového týmu, která se odráží na hodnotě produktu. Je to člověk, který odpovídá za správu backlogu. Určuje práci vývojovému týmu. **Vývojový tým** se skládá z profesionálů, kteří dodávají přírůstek produktu na konci každého sprintu. Tým si sám organizuje práci. Optimální velikost týmu je 3-10 členů. Tým je vhodné soustředit na jednom místě, kvůli komunikaci mezi členy týmu, která je základem úspěchu. Je možné používat moderní styl komunikace, která umožňuje použití SCRUM i po světě. **SCRUM master** odpovídá za osvojení a dodržování pravidel SCRUMU. Trvá na uplatnění technik a pravidel SCRUM. Slouží lidem ve vývojovém týmu a pomáhá jim změnit chování, aby byla práce co nejefektivnější. [21]

Obrázek 5. – Scrum

Zdroj: http://www.agilenutshell.com/scrum

Činnosti SCRUM. Všechny činnosti jsou časově ohraničené. Pokud je definována např. délka sprintu, nemůže být jeho délka zkrácena nebo prodloužena. Ostatní činnosti mohou skončit, když dojde k dosažení jejich cíle. Základní jednotkou vývoje je takzvaný **Sprint.** Sprint je jasně definovaný časový úsek většinou několik týdnů. Sprint je souhrnem předepsaných činností. Každá činnost slouží jako kontrola a adaptace některého aspektu SCRUM. Sprinty obvykle mají shodnou délku, jakmile jeden sprint končí, okamžitě začíná další. Každý sprint je započat plánovaným setkáním, kde jsou definované jednotlivé úkoly. Každý den sprintu je zahájen denní schůzkou. Sprint je ukončen setkáním se zpětným zhodnocením provedené práce a definováním vylepšení práce pro následující sprint. Výsledky sprintu jsou předkládány zadavateli. [18]

Artefakty SCRUM reprezentují práci a její hodnoty. Jejich vlastností je maximální transparentnost klíčových informací. **Produktový backlog** je seznam všeho, co může být v produktu potřeba. Postupem času se vyvíjí. V úvodních fázích obsahuje jen základní požadavky. Dále se přidávají popisy vlastností, funkcí, požadavků, rozšíření a oprav chyb, které budou v produktu provedeny. Dalším artefaktem je **Backlog sprintu**. Je to množina úkolů vybraných z produktového backlogu včetně plánu dodání produktového přírůstku a splnění cíle sprintu. Ukazuje všechnu práci nutnou k ukončení daného sprintu. Je velmi podrobný a odráží práci vývojového týmu na denní bázi. Posledním artefaktem je

Přírůstek. Jedná se o sumu dokončených položek produktového backlogu v průběhu aktuálního sprintu a všech předchozích sprintů.

1.7.5 Extrémní programování (XP)

Jedná se o agilní metodiku vývoje software. Její počátky sahají do počátku 90. let. Prvně definována a popsána je v knize Kenta Becka, Extreme Programming Explained z roku 1999. Tato metodologie předepisuje jednotlivým účastníkům vývojového procesu tradiční činnosti, které jsou však dovedeny do extrému. Oproti tradičním metodikám, které dodávají konečný produkt ve vzdáleném bodu v budoucnu, extrémní programování předkládá zákazníkovi software, který potřebuje ve formě a čase, kdy sám zákazník potřebuje. Podporuje vývojáře, aby byli schopni reagovat na měnící se zákaznické požadavky i v pozdní fázi vývojového cyklu. Extrémní programování je založeno na týmové práci. Všichni účastníci (manažeři, zákaznici a vývojáři) jsou rovnocennými partnery. Tým si sám organizuje práci, jak nejlépe uzná za vhodné s cílem co nejrychlejšího vyřešení problému. Tým je zasazen do jednoduchého efektivního prostředí, které napomáhá produktivitě. [19]

Proč extrémní? Osvědčené činnosti vývoje jsou dovedeny do extrémů. Např. pokud se ukázala jednoduchost programu jako přínosná, pak v extrémním programování je program držen na co nejmenší úrovni složitosti. Vždy je programováno, co je nezbytné, aby byla zachována funkčnost.

a) **jednoduchost** – systém je ponechám v nejjednodušší podobě, která vyhovuje požadavkům funkčnosti,
b) **revize** – pokud se osvědčí revize a kontrola zdrojového kódu, je prováděna revize a kontrola neustále, např. za pomocí tzv. párového programování. Párové programování je technika, kdy dva programátoři píší kód na jednom počítači,
c) **návrh** – pokud se osvědčí návrh, budou všichni neustále navrhovat a vylepšovat současný návrh,
d) **architektura** – v případě důležitosti architektury, bude neustále vylepšována a přepracovávána architektura,

e) **testování funkcionality** – pokud se osvědčí testování, všichni budou průběžně testovat.
f) **krátké iterace** – snaha o co nejmenší jednotlivé kroky. [14]

XP je metodika vhodná pro malé až střední týmy (od dvou až po deset programátorů). Programátoři se musí neustále potýkat s rychle měnícím se zadáním. Jediným exaktním, jednoznačným, změřitelným, ověřitelným a nezpochybnitelným zdrojem informací je zdrojový kód. [20]

Metodologický rámec Extrémního programování tvoří čtyři základní hodnoty a jedna hlubší hodnota - respekt:

a) **komunikace** – je základním předpokladem fungování metodiky, v případě problémů v projektu, je s velkou pravděpodobností chyba v zanedbání komunikace,
b) **jednoduchost** – snaha o tvoření co nejjednodušší nejmenší části, která zachovává funkčnost, v případě potřeby je vždy možné ji rozšířit,
c) **zpětná vazba** – předává informace o aktuálním stavu vývoje a o všech záležitostech, které se týkají vývoje např. testování,
d) **odvaha** – důležitý předpoklad, bez kterého není možné extrémní programování aplikovat,
e) **respekt** – každý člen týmu se musí angažovat, zajímat se o práci ostatních a nepracovat samostatně.

Programátoři neustále komunikují se svými kolegy programátory a zákazníky. Design produkt je držen na co nejjednodušší úrovni. Od zákazníků neustále přichází zpětná vazba a testování probíhá již od raných fází vývoje. Dodání softwaru pro zákazníky včetně implementace navržených změn probíhá co nejdříve. S tímto přístupem je tým extrémního programování schopen odvážně reagovat na měnicí se požadavky a technologie. Základem extrémního programování jsou jednotlivé zdánlivě jednoduché činnosti, které dávají smysl, až pokud jsou všechny aplikovány zároveň. [19]

Na základě metodologického rámce bylo popsáno dvanáct základních postupů extrémního programování, které slouží k vytvoření kvalitního produktu:

a) **plánovací hra** – ve spolupráci zákazníka a vývojového týmu je určeno, jak dosáhnout výsledku co nejefektivněji. Zákazník uvede seznam funkčních požadavků. Tyto požadavky jsou rozepsány do uživatelských scénářů. Vývojový tým je následně zhodnotí, jak rychle a za jakou cenu je schopen jednotlivé funkce vytvořit. Zákazník na základě potřeb vyhodnotí, co je pro něj podstatné a co má být vyvinuto.
b) **zákazník na pracovišti** – při vývoji je přítomen zákazník (uživatel), který určuje požadavky a stanovuje priority. Zákazník se stává součástí vývojového týmu,
c) **malá verze** – uvolňování malých verzích v co nejmenší konfiguraci. Četnost uvolňování je závislá na velikosti výsledné aplikace,
d) **metafora** – celý postup vývoje je převeden na metaforu o tom, jak má celý systém fungovat. Výhoda spočívá v jednoduchém pochopení všemi zainteresovanými subjekty,
e) **jednoduchý návrh** – protože se předpokládá s měnícími se požadavky, je vždy vyvíjen jen nezbytný základ,
f) **testování** – testování je základ úspěchu XP, všechny testy musí proběhnout úspěšně.
g) **refaktorizace kódu** – pomocí refaktorizace programátoři kontrolují, jestli se nedá kód vylepšit nebo zjednodušit. Tato činnost napomáhá k odstranění duplicit a zvyšuje čitelnost kódu. Společné vlastnictví umožňuje každému programátorovi provádět změny. Díky standardizaci je kód čitelný a neustálé testování zachovává funkčnost,
h) **párové programování** – jeden zdrojový kód píše dvojice programátorů. Ve dvojici se role dělí na jednoho, který píše kód a druhý, který přemýšlí o souvislostech přidávaného kódu. Role se průběžně střídají. Díky neustálé komunikaci v podobě zpětné vazby, je zabráněno množství chyb,
i) **standardy kódu** – kód vyžaduje dodržování určených standardů, aby byla umožněna efektivní práce kohokoliv na jakémkoli kódu, jedná se o podmínku pro párové programování a refaktorizaci kódu,
j) **tempo** – je nutné nastavit udržitelné tempo. Je dokázáno, že dlouhodobé přetěžování lidí vede k chybám a ztrátě efektivity,
k) **společné vlastnictví** – jedná se o další podmínku pro refaktorování a společné programování. Neexistuje individuální vlastnictví jednotlivých částí kódu,

l) **průběžná integrace** – do celku jsou integrovány nové funkce co nejčastěji, alespoň jednou denně. Po každé integraci následují testy, který odhalí případně chyby a jejich oprava je okamžitá. [19]

1.8 Metodiky používané pro vývoj informačních systémů

Základní přístupy, metodiky a jak fungují jednotlivé modely, byly popsány výše. V této podkapitole se zaměřím na to, jaké metodiky se používají přímo při vývoji informačních systémů.

Ve vývoji informačních systémů se používají se dva hlavní metodické přístupy. Tyto přístupy jsou rigorózní a agilní. Tyto metodiky pracují s rozdílnými předpoklady a s odlišným pohledem na vývoj software. Rigorózní a agilní metodiky představují dvě skupiny metodik, které vycházejí z odlišných dispozic a různého přístupu k vývoji software.

Rigorózní metodiky chápou vývoj jako sérii po sobě jdoucích úkolů a jsou založeny na sériovém (vodopádovém) vývoji. Předpokladem rigorózních metodik je přesvědčení, že procesy při vytváření informačních systémů lze popsat, plánovat, řídit a měřit. Můžeme se setkat i s rigorózními metodikami založenými na iterativním a inkrementálním vývoji.

Rigorózní metodiky:

a) model zralosti SW,
b) metodika OPEN,
c) metodika Rational Unified Process,
d) metodika Enterprise Unified Process.

Agilní metodiky se odprošťují od pěvně daného systému rigorózních metodik a následují trendy ve změnách technologického a ekonomického prostředí, ve kterém je kladen velký důraz na rychlé zavádění nových systémů a změn. Agilní metodiky umožňují rychlou a pružnou reakci na současný vývoj. Všechny agilní metodiky sjednocuje myšlenka, že jedinou cestou, jak prověřit správnost navrženého systému, je vyvinout jej co nejrychleji, představit zákazníkovi a provést požadované změny.

Agilní metodiky:

a) Dynamic Systems Development Method (DSDM),
a) Adaptive Software Development (ASD),
b) Feature–Driven Development (FDD),
c) Extrémní programování (XP),
d) Lean Development,
e) Scrum,
f) Crystal metodiky,
g) Agilní modelování (Agile Modeling). [14]

1.9 Prostředí SAP ERP

Název společnosti SAP je tvořen ze zkratek slov „Systémy Aplikace a Produkty v oblasti zpracování dat“. Její historie sahá do roku 1972 a v dnešní době má zastoupení ve více než 130 zemích světa a stala se předním dodavatelem podnikových aplikací, které pomáhají provozovat firmy na celém světě. Mimo podnikových aplikací nabízí také produkty z oblasti databází, analytiky, a nově cloudu a mobilních zařízení. Orientuje se jak na velké, tak i na malé a střední firmy

V této práci se soustředím na základní popis informačního systému SAP ERP. Tento systém pokračuje jako nástupce systému SAP R/3 z roku 1992 (číslo 3 označuje databáze, aplikační server a klient). Poslední verze SAP ERP je 6.0 představená v roce 2013.

Informační systém SAP umožňuje automatizaci běžných činností a procesů v prostředí podniku. Pokrývá však i mimopodnikovou sféru procesů. S pomocí informačního systému SAP je podnik schopen efektivně v rámci jediného systému elektronicky zpracovávat finanční dokumenty, manipulovat s položkami na skladě, vyplácet mzdy, efektivně komunikovat s odběrateli a dodavateli atd. Jako většina informačních systém je SAP centralizován, což umožňuje optimalizovat náklady na provoz a údržbu. Propojuje většinu částí podniku a umožňuje zvyšovat efektivitu procesů.

Systém funguje na síťové architektuře klient – server. Architektura odděluje klienta s grafickým rozhraním a server, kde dochází k samotné práci s daty. Ke komunikaci se

využívá počítačová síť. Výhodami jsou centrální správa dat, datová integrita, snížení zátěže na datovou síť a jiné. Samozřejmostí je víceuživatelský přístup a odstranění konfliktů při souběžné práci s daty.

Změny zasahující do fungujícího systému podniku můžou negativně ovlivnit ostatní procesy a ohrozit fungování běžných činností podniku. Pro minimalizaci rizika nechtěného zásahu je systém rozdělen do tří vedle sebe běžících instalací. Mezi těmito instalacemi je zajištěno předávání dat. Tyto jednotlivé instalace jsou systém Vývojový, Testovací a Produktivní. Vývoj a konfigurace sw probíhá nejprve ve vývojovém systému. V prostředí SAP je možné využívat všechny metodiky využívané pro vývoj software uvedené výše. Po ukončení vývoje je sw otestován v testovacím systému, který obsahuje kopii skutečných dat z produktivního systému. Po úspěšném otestování je konfigurace aplikována na systém produktivní. Předávání dat se nazývá transport a datová spojení mezi systémy se označují jako transportní cesty. [17]

1.9.1 Struktura uživatelů

a) **Koncoví uživatelé** – provádí běžné činnosti v systému. Například zpracování faktur, evidence zásob, zpracování mezd atd.
b) **Klíčoví uživatelé** – jsou zkušení uživatelé, kteří zastupují jednotlivé oblasti v business procesech. Odpovídají za správný běh aplikací a rozdělování práv uživatelům v daných oblastech. Jsou odpovědní také za hlášení chyb, dávají podněty k vylepšení systému a provádí testování. Fungují jako mezičlánek komunikace mezi business oblastí a IT podporou.
c) **Systémoví organizátoři –** jedná se o zkušené pracovníky IT, kteří mají hlubší znalosti a zkušenosti s fungováním business procesů a samotného systému SAP. Starají se o provádění změn a opravu chyb. Realizují také implementační projekty a konfiguraci systému. [17]

1.9.2 Moduly SAP ERP

SAP ERP rozděluje systém na řadu modulů, které korespondují se základním rozdělením podniku do jednotlivých oddělení. Organizace využívají pouze některé. Mezi hlavní moduly patří:

a) FI (Financial Accounting),
b) CO (Controlling),
c) IM (Investment Management),
d) SD (Sales and Distribution),
e) MM (Material Management),
f) PP (Product Planing),
g) PM (Plant Maintenance),
h) QM (Quality Management),
i) HR (Human Resources).

2 Možnosti kontroly vývoje software

Kontrolou obecně označujeme činnosti jako přezkoušení, ověření, testování, přezkoumaní nebo revize. Poskytuje informaci v podobě zpětné vazby o stavu zkoumaného jevu nebo objektu. Ověřuje, zda skutečný stav se rovná žádoucímu stavu a zda existují opatření k dosažení souladu mezi skutečným stavem a stavem žádoucím. Na základě kontroly se můžou provést akce, které zajistí nápravu případné neshody s žádoucím stavem a bude dosaženo určených cílů. V podnikové praxi to může znamenat minimalizaci nákladů.

2.1 Reporting

Reporting je proces, který má na starosti sběr a podávání přehledových zpráv o aktuální stavu sledovaného jevu. Jedná se o důležitou činnost controllingu, která má nad reportingem kontrolu a odpovědnost. Jeho cílem je poskytnout informace. Reporting neslouží jen k zpracování a reprezentaci dat, ale také jako centrum uchovávání informací. Tyto informace je dále možné sledovat v čase. Dále slouží pro tvorbu statistik a různých analýz. Reporting je běžně relativně samostatná součást celého informačního systému. Controlling můžeme dělit na základě frekvence a pravidelnosti jednotlivých reportů. První kategorií je standardní reporting, který probíhá v přesně daných intervalech např. den, týden, měsíc, rok nebo více. Do druhé kategorie spadá mimořádný reporting. Provádí se nahodile při různých příležitostech např. audit. [22]

Informace musí být správně interpretovány v závislosti na osobě, která informace přijímá. Uživatelů může být mnoho s různými požadavky, kteří kladou obsahový i formální důraz na zpracování dokumentu. Cílové skupiny reportingu můžeme dělit na dvě skupiny, interní uživatelé a externí adresáti. Mezi interní uživatele patří vlastníci a management, např. v případě akciových společností se jedná o představenstvo a dozorčí radu. Tito adresáti mají rozhodovací pravomoci a odpovídají za výsledky činností. Externí adresáti jsou druhá a mnohem obsáhlejší skupina. Patří mezi ně například:

a) zaměstnanci podniku, ti tvoří vnitřní zájmovou skupinu. Jejich zájmem není pouze prosperita a dobré jméno podniku, což jsou vlastnosti interních uživatelů, ale také

sledují svůj vlastní zájem v podobě maximalizace mezd. Vysoké mzdy vedou k vyšším nákladům firmy,

b) spolupracující podniky, odběratelé, dodavatelé, banky atd.,
c) státní orgány pověřeny výkonem kontrolních a jiných funkcí ve vztahu k činnostem podniku, finanční úřad, pracovní úřad a jiné,
d) orgány veřejné správy jako jsou krajské úřady, zastupitelstva obcí a měst atd.
e) široká veřejnost, společenské organizace a jiná uskupení.

Na tomto výčtu cílových skupin je patrné, že není možné jednou formou uspokojit všechny požadavky.

Reporting je úzce svázán s používaným informačním systémem podniku. Různý informační systém vytváří rozdílné podmínky pro kvalitní reporting. Systémy typu ERP, vytváří vhodné prostředí pro tvorbu reportingu. Pro zlepšení kvality je možné použít nadstavbové systémy součástí Business Intelligence jako je např. EIS (Executive Information Systems). Při implementaci reportingu je dobré dodržovat základní zásady:

a) identifikace uživatelských zpráv a analýza jejich požadavků,
b) odlišení zpráv dle potřeb uživatelů na interní a externí,
c) zvolení vhodné formy reportů (elektronická, tištěné zprávy),
d) vybrání vhodné distribuce zpráv,
e) využívání zpětné vazby, která následně slouží k vylepšení reportingu. [23]

2.2 Metriky, Ukazatele výkonnosti (PI, KPI)

Jak uvádí učební text Softwarové inženýrství: „*Pro hodnocení přínosů a efektivnosti IS používáme měření (formální popis vlastností zvoleného výseku světa). V IS/IT se setkáváme spíše s pojmem metrika, což je z pohledu statistiky pojem nesprávný. Funkce, která zkoumaným entitám (objektům či jevům) v reálném světě přiřazuje čísla, se nazývá míra a číslo přiřazené dané entitě pak hodnota míry. Měření převede zkoumaný výsek empirického světa na nějakou formální strukturu. Z ní pak pomocí různých metod můžeme odvodit nové poznatky o zkoumaném výseku světa (tzv. interpretace).“* [24]

Do základní kategorie ukazatelů (metrik) patří:

a) **tvrdé metriky** – jsou objektivně a snadno měřitelné, např. rychlost, cena,
b) **měkké metriky** – není možné objektivně měřit např. subjektivní hodnocení, možné využívat číselných stupnic.

Dále můžeme rozlišovat metriky na:

a) **finanční** – návratnost investic, cena jedné transakce, cena změny nebo chyby, hodnota celkového vlastněného ICT tzv. Total Cost of Ownership (TCO),
b) **nefinanční** – mezi ně patří velmi důležitý ukazatel přínosů IT, produktivita (poměr mezi vstupními náklady a výstupním užitkem), zkrácení doby vývoje, snížení počtu reklamací atd. [24]

Příklady ukazatelů kvality používaných při hodnocení software jsou ukázány u modelu FURPS. Pro činnost kontroly je důležité nastavit vhodné údaje, které budou následně kontrolovány. Ukazatel výkonnosti je překlad z anglického termínu Performace Indicator. Pojem Klíčový ukazatel výkonnosti vznikl překladem anglického termínu Key Peformance Indicator (KPI). Tento pojem označuje indikátory, ukazatele nebo metriky výkonnosti. KPI se od PI liší svoji důležitostí. Je nasazován na měření výkonnosti procesů, služeb, organizačních útvarů nebo celé organizace. Používají se na všech úrovních organizace. Vyjadřují požadovanou výkonnost např. kvalitu, efektivnost, rychlost, hospodárnost.

Použití a implementace KPI není nijak složitá. Hlavní problém leží v identifikace jaký ukazatel sledovat. Správné zvolení ukazatele se odvíjí od schopnosti odhalit, co je pro organizaci důležité. Jaký údaj sledovat vychází z pochopení, zvládnutí a zkušenosti s jednotlivými procesy. KPI musí být nastaven tak, aby reálně měřil sledovanou aktivitu a napomáhal dosahovat stanovené cíle.

V knize Key Peformance Indicators David Parmenter definoval sedm vlastností efektivních klíčových ukazatelů výkonnosti:

a) **nefinančně zaměřené** – neměřit peněžní vyjádření,
b) **včasné** – měření by mělo probíhat pravidelně a v ten „správný čas“,
c) **zaměřené na vedení** – ukazatele by měly být takové, aby na ně mohlo reagovat vedení podniku,

d) **jednoduché** – ukazatele by měli být jednoduché a srozumitelné všem, aby mohli případně reagovat a podílet se na nápravě,
e) **týmové** – odpovědnost by měla být dána skupině pracovníků,
f) **významné** – nemá smysl měřit hodnoty, které nevyjadřují důležité informace,
g) **povzbuzující** – ve smyslu k provádění navazujících činností. [25]

Při identifikaci KPI můžeme využít kritéria SMART k nastavování cílů. Zkratka znamená Specific (konkrétní význam pro zainteresované osoby), Measurable (měřitelné), Achievable (dosažitelné), Realistic (realistické), Time-related (aktuální a vztažené k časovému úseku). Jak je uvedeno, identifikování vhodného KPI je zásadní. Liší se dle prostředí. KPI ve školství například může ukazovat počet neúspěšných žáků. V prostředí obchodu např. procentuální marže při prodeji. Při vývoji software je nutné uplatnit specifické ukazatele.

2.3 Kvalita Software

Software Quality Assurence se skládá z monitorování procesů vývoje software za účelem zajištění kvality. Existuje mnoho používaných metod. Některé metody mohou podporovat plnění jednoho nebo několika standardů např. ISO 9000 nebo model CMMI. SQA pokrývá celý životní cyklus vývoje software, který zahrnuje procesy jako, definice požadavků, design, kódování, revize kódu, testování, uvolňování, integrace a údržba.

Tým zajišťující QA se neúčastní v samotném procesu vývoje software, ale kontroluje, jestli se proces řídí určenými příručkami a standardy. Sleduje, jestli jsou standardy, procesy a procedury pro projekt vhodné a kontroluje jejich implementaci. Zodpovídá za sběr a prezentaci vyhodnocení nastavených metrik.

Software Quality Control (SQC) oproti SQA se účastní přímo procesu vývoje software. Přímo zajišťuje kontrolu, zda jsou standardy v dokumentaci navržené týmem SQA fakticky dodržovány. SQC definuje a provádí testování software. [26]

QA a QC byly převzaty z tradičního výrobního světa. Jak je uvedeno v kapitole jedna, tradiční výroba (fyzického výrobku) a výroba software se výrazně liší. Protože software je v podstatě nehmotný, jeho vlastnosti se špatně měří a posuzují. Aplikace QA a QC

na vývoj software se setkala s problémy. Aby se předešlo těmto problémům, byly definovány parametry posuzující vlastnosti software. Tyto parametry jsou označovány jako atributy kvality, metriky a funkční požadavky. Mezi hlavní používanou definici atributů kvality patří FURPS model od Roberta Gradyho a společnosti Hewllet-Packard. FURPS posuzuje kvalitu software na základě pěti základních hledisek:

a) **funkčnost** – popisuje hlavní činnost programu. Definuje, co produkt děla a jak to dělá. Zda naplňuje požadavky zákazníka atd.,
b) **použití** – hodnotí míru použitelnosti produktu. Zahrnuje aspekty jako ovladatelnost, vzhled prostředí, dokumentaci a zpětnou vazbu. Jedná se o zákaznický pohled,
c) **spolehlivost** – zahrnuje aspekty jako dostupnost, přesnost a schopnost zotavení po selhání. Hodnotí četnost a závažnost chyb. Pro vyjádření spolehlivosti se používá např. metrika MTBF (střední hodnota doby mezi chybami),
d) **výkon** – měří technické parametry software. Mezi ně patří rychlost odezvy, doba trvání klíčových funkcí, vytížení zdrojů, zatížení síťového provozu atd.,
e) **podpora** - definuje možnosti údržby a podpory software např. možnost testování, přizpůsobení, kompatibilita, nastavení, lokalizace, rozšíření a další. [27]

FURPS model se v poslední době rozšířil kvůli nedostačujícímu počtu kategorií. Model se rozšířil o design, implementaci, rozhraní, systémové požadavky apod. Označuje se jako FURPS+ a je v současnosti používán v praxi.

CMMI – je model organizace práce určený pro vývojové týmy. CMMI je zkratka ze slov Capability Maturity Model Integration. CMMI byl vyvinut experty z průmyslu, vlády a Software Engineering Institute (SEI) na univerzitě Carnegie Mellon. Model poskytuje návod pro vývoj nebo zlepšovaní procesů, zejména z oblasti vývoje software. Oproti modelu ISO 9000:2001 je možné model CMMI přímo určen na konkrétní obor, v případě CMMI-DEV na vývoj software. Existují ještě další varianty např. CMMI-SVC a CMMI-ACQ. CMMI Definuje stupňovitý model zralosti o několika úrovních:

a) **počáteční (Initial)** – procesy nejsou ve firmě naplno vykonávány, není je možné předvídat ani řídit,
b) **řízená (Managed)** – procesy a projekty je možné řídit a plánovat,

c) **definovaná (Defined)** – procesy jsou definovány,
d) **kvantitativně řízená (Quantitatively Managed)** – procesy je možné měřit a kontrolovat,
e) **optimalizující (Optimizing)** – zaměřené na zlepšení procesů. [38]

2.4 Testování

Testování patří do podmnožiny procesu ověřování kvality. Podle metodik vývoje software je možné pozorovat, že testování výrazně zasahuje do životního cyklu vývoje. Metodik je mnoho a liší se zaměřením, rozsahem, přístupem k řešení a také přístupem k testováním. Předmětem testování je vyhledávání chyb. Cílem testování však není chyby pouze vyhledat, ale co nejdříve zajistit jejich nápravu. S tím souvisí sledování chyby po celou dobu jejího životního cyklu od nalezení a ohlášení po nápravu. Testování je proces, při kterém dochází k ověření, zda software odpovídá stanoveným požadavkům. [29]

Definice dle Kanera [32] popisuje testování jako empirický technický výzkum kvality testovaného produktu nebo služby, prováděný za účelem poskytnutí těchto informací stakeholderům. Roudenský definuje testování jako: *„Proces řízeného spouštění softwarového produktu s cílem zjistit, zda splňuje specifikované či implicitní potřeby uživatelů.“* [33] Od starších definic se tato definice liší odklonem od čistého testování softwaru za účelem nalezení chyb, avšak stále obsahuje metody statického testování, které obsahuje zkoumání dílčích produktů – produktů, zdrojového kódu atp. V dnešní době testování slouží jako nástroj k získávání informací zda produkt vyhovuje zamýšlenému použití. Při tom připouští existenci chyb. Všechny definice mají společný prvek a to je cíl zajištění kvality software. Testování samotné nezajistí kvalitu produktu, pouze poskytuje informace. Za zajištění kvality stojí procesy QA, které je popsáno v kapitole 2.4. Testování může být nekonečný proces a o jeho ukončení rozhoduje oprávněná osoba na základě různých faktorů např. snižování efektivity nebo časová omezení. Podléhá kompromisu mezi kvalitou, časem a rozpočtovým omezením.

Posun v cílech testování je možné sledovat na 5 fázích vyspělosti testovacího procesu (z pohledu mentality testera) podle Beizera:

a) **Úroveň 0:** Mezi testováním a laděním (debuggingem) není rozdíl. Testování se příliš neodlišuje od ladění. Spolu s ním se snaží o odstranění defektů,
b) **Úroveň 1:** Smyslem testování je prokázat, že software funguje. Testování slouží pro předvedení funkčnosti software. Zaměřuje se na funkčnost a její demonstraci,
c) **Úroveň 2:** Smyslem testování je prokázat, že software nefunguje. Testovaní slouží pro nalezení chyb. Zaměřuje se na destrukci,
d) **Úroveň 3:** Smyslem testování není prokazování ničeho konkrétního, ale snížení rizika. Testování je rozšířeno mimo testování kódu. Zaměřuje se také na požadavky a je součástí vývojového cyklu software,
e) **Úroveň 4:** Testování je duševní disciplína vedoucí k produkci softwaru s nízkým rizikem. Testování se spolu s řízením kvality snaží předcházet vzniku chyb v požadavcích, návrhu a kódu. [31]

Testování s celým řízením kvality by mělo být zahájeno co nejdříve je to možné. Pokud je případný problém odstraněn již při identifikaci požadavků, představuje jeho odstranění menší úsilí a mnohem nižší prostředky, než při odhalení problému až při představení produktu. Včasné odhalení problému koresponduje s velikostí nákladů na jeho opravu.

2.4.1 Role v testovacím týmu

Testování větších projektů neobstarává pouze jeden pracovník. Na proces testování jsou nasazovány celé týmy. Dle velikosti projektu a rozsahu testování se velikost týmu mění. Je možné se setkat s těmito rolemi:

a) Tester – provádí samotné testování dle předem připravených testovacích scénářů od analytika testování. V případě nalezených chyb je eviduje do bug trackingu. Výsledek testu reportuje. V případě opravy chyby je jeho úkolem znovu otestovat. Správný tester by se měl svým uvažováním lišit od běžného uživatele. Jeho úkolem je prověřit všechny vlastnosti a funkce produktu. K tomu aby prověřil všechny možné scénáře, si tester klade otázky typu „co se stane když?“.
b) Analytik testování – se zabývá tvorbou test analýzy. Snaží se zjistit o aplikaci co nejvíce informací, aby navržené testy pokryly všechny klíčové funkcionality. Po

nastudování maximálního množství informací připravuje dokumenty pro testování. Patří sem příprava testovací scénáře a jednotlivé testovací případy.

c) **Vedoucí testování –** odpovídá za testovací tým. V případě větších projektů může být více testovacích týmů. Úlohou vedoucího testování je vytvoření testovacího plánu v souladu se strategií testování, kontrola a korektivní akce. Dále zadává úkoly jednotlivým testerům a spravuje systém pro hlášení chyb. Vlastností vedoucího testování musí být samotné testování a znalost aplikace, navíc zkušenosti s řízením lidí a organizační dovednosti.

d) **Manažer testování –** odpovídá ze celý projekt testování. Je nejvýše postavený v hierarchii testovacího týmu. Jeho činnost se skládá z určení přístupu k testování, stanovení cílů testování, kritérií produktu, jednání s managementem, zákazníkem i vedoucím vývoje, koordinování akceptačního testování, spolupráci s QA atd. Rozhoduje také o skladbě testovacího týmu. Při menším projektu bývá tato role sloučena s rolí vedoucího testování. [31]

2.4.2 Kategorie testů

Úkolem vedoucího testování a manažera testování patří určení přístupu k testování, stanovení cíle atd. Je to první předpoklad pro správné testování. Nejprve je nutné rozhodnout, jaké druhy testů se použijí, jaké použít nástroje a kdy je použít. Prvním dělením testování je na testování **statické a dynamické.** Statické testování se zabývá zkoumáním dílčích částí software např. procházení kódu. K jeho provádění není nutné aplikaci spouštět, a proto je možné začít se statickým testováním již v raných fázích vývoje. Oproti statickému testování dynamické testování vyžaduje funkční prototyp software. Uplatnění dynamického testování najdeme více v pozdějších fázích vývoje.

Černá a bílá skříňka se řadí do rozdělení testování dle přístupnosti ke kódu a vnitřních funkcí programu. Při použití testování černé skříňky není možný přístup ke kódu. Program je jakoby uzavřen do černé skříňky. Zkoumá se pouze chování programu. Sledujeme pouze výsledek po zpracování vstupních dat. Do této kategorie patří testy uživatelského rozhraní, akceptační testy a testování dle scénářů. Testovaní typu **bílé skříňky** (říká se ji také skleněná) probíhá s tím, že tester má plný přístup ke zdrojovému kódu. Chování programu

může sledovat jak na uživatelské úrovní, tak také vidí, jak program uvnitř pracuje a může lépe odhadnout, kde chyby hledat. Třetím, spíše okrajovým typem, je takzvaný testování šedé skříňky. Jsou známy pouze základní informace o vnitřním chování, ale nejsou úplné, tudíž se nejedná o testování bílé skříňky.

Dalším rozdělením je rozdělení na **automatické a manuální testování**. Liší se podle toho, jestli je provedeno člověkem nebo strojem. Při testováním, které se opakuje a vyžaduje zpracování velkého objemu dat, jako například zátěžové testování, je vhodnější automatické testování. Při potřebě zásahu člověka v podobě hodnocení, různého přístupu atp. je lepší použít manuální testování. Při tvorbě automatického testování je nejprve nutné napsat testovací skript. Vytvoření vyžaduje čas. Proto volba mezi manuálním a automatickým testováním také závisí na časové náročnosti a velikostí nákladů.

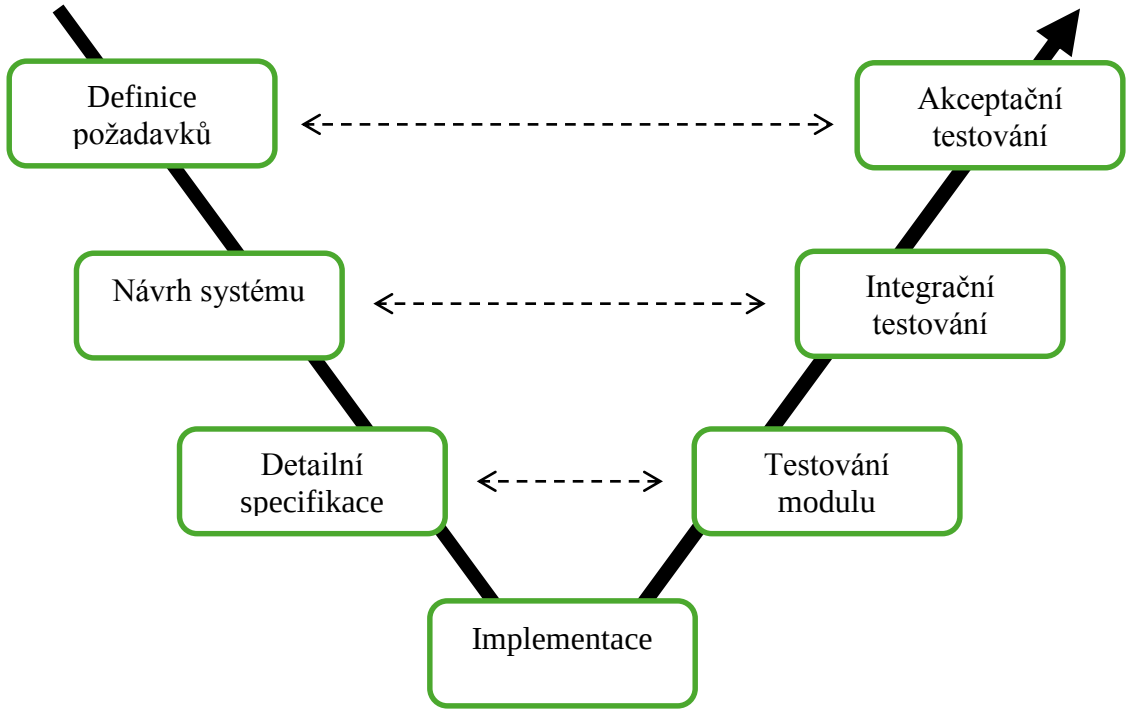
Podle úrovně, fáze a délky časového odstupu testování od napsání kódu se testování dělí do pěti základních úrovní.

a) **testování programátorem** – provádí sám programátor po napsání kódu. Při samotném vývoji programátor kontroluje, co napsal. Odhalí tím, jestli kód splňuje to, co programátor chtěl. Navíc kontroluje i samotnou strukturu kódu, jestli kód funguje správně, tím může odhalit chyby, jejichž oprava je v této fázi nejméně nákladná.
b) **testování jednotek** – je proces testování co nejmenších částí kódu. Testovanou jednotkou se rozumí samostatná testovatelná část aplikačního programu. Testovanou jednotkou může být v procedurálním programování program, funkce, procedura proměnná atd. v objektovém programování je jednotkou obvykle třída nebo metoda. Snahou je oddělit testovanou část programu od ostatních částí. Za tímto účelem je možné vytvářit pomocné objekty, které pouze simulují chování a vlastnosti ostatních částí.
c) **funkční testování** – se provádí na straně dodavatele. Kontroluje se, jestli aplikace plní všechny úkoly, pro které je určena. Ověřuje se správná funkčnost, a jestli program odpovídá požadavkům zákazníka. Funkční testy jsou podmínkou pro vstup do fáze systémové testování.
d) **integrační testování** – Cílem integračních testů je ověřit bezchybnou komunikaci mezi jednotlivými komponentami uvnitř aplikace. Toto testování je označováno za

vnitřní integraci. Netestuje se integrace pouze mezi komponentami uvnitř aplikace, ale také mezi komponentou a operačním systémem, hardwarem či rozhraním různých systémů (vnější integrace). Aplikace sama o sobě může fungovat bezchybně, ale pokud nedokáže komunikovat s okolím, není dobře využitelná. Testování se provádí postupně od dvou komponent. Po úspěšné otestování se přidávají další komponenty. Integrační testování není nezbytné pro výslednou bezporuchovost aplikace, avšak čím dříve je chyba odhalena, tím více klesá čas a náklady na její nápravu. Integrační testy je možné provádět automatizovaně.

e) **systémové testování** – přichází na řadu v pozdějších fázích vývoje. Slouží jako výstupní kontrola produktu před předáním zákazníkovi. Aplikace je pomocí systémového testování ověřována jako celek. Ověřuje se reálné chování aplikace s představami zákazníka. Testování probíhá dle připravených scénářů obvykle v několika kolech. Chyby nalezené v jednotlivých kolech jsou opraveny a testovány znovu v dalším kole. Systémové testování patří k nejdůležitějšímu, bez jeho provedení by byla významně ohrožena výsledná kvalita aplikace.
f) **akceptační testování** - je prováděno zákazníkem aplikace. Testuje se splnění všech požadavků. Testování probíhá v testovacím prostředí na straně zákazníka dle připravených scénářů, které připravil zákazník ve spolupráci s dodavatelem. Při objevení chyby hraje klíčovou roli komunikace mezi zákazníkem a dodavatelem. Chyba musí být nahlášena a opravena v co nejkratším čase. [32]

V-model ukazuje vztah mezi různými fázemi vývoje a následně jejich testováním. Levá část modelu uvádí fáze vývoje a pravá druhy testování. Nejdříve testování probíhá od malých částí aplikace a postupně přechází v testování celé aplikace. Přechod mezi fázemi je podmíněn úspěšným splnění předchozí fáze.



Obrázek 6. – V-model

Zdroj: Clarus Concept of Operations – vlastní zpracování [34]

2.4.3 Testovací dokumentace

K testování samotnému patří také dokumentace. Bez dokumentace by se celé testování mohlo potýkat se ztrátami informací. Bez dokumentace není možné definovat postupy pro testování. V malých projektech je možné držet dokumentaci z části na neformální úrovni, ale nemělo by chybět vymezení, co a jak se bude testovat, včetně zaznamenávání a reportů chyb. Ve větších projektech s velkým týmem lidí se bez dokumentace testování neobejde. Pro tvorbu detailní testovací dokumentace s přesně definovaným obsahem existují metodiky a standardy. Nejznámější je standard IEEE-829 Standard for Software and Systém Test Documentation. Definuje osm fází testování softwaru a systému. Výsledkem jednotlivých fází je jiný typ dokumentu. IEEE-829 definuje pouze strukturu dokumentů. Obsah dokumentů není nijak ve standardu upřesněn a není vyžadováno ani využití všech dokumentů. Tyto dokumenty jsou:

a) Master Test Plan (MTP),
b) Level Test Plan (LTP),

c) Level Test Design (LTD),
d) Level Test Case (LTC),
e) Level Test Procedure (LTPr),
f) Level Test Log (LTL),
g) Anomaly Report (AR),
h) Level Interim Test Status Report (LITSR),
i) Level Test Report (LTR),
j) Master Test Report (MTR). [35]

Rozhodnutí o formě dokumentace a jejím vedení dávají vývojové a testovací týmy ve spolupráci se zákazníkem. Mimo jiné může být sledováno, kolik bude tvorba dokumentace stát. Sledováno je např. kolik stojí tvorba dokumentace na jeden testovací případ nebo jeden případ užití.

Mezi nejpoužívanější dokumenty patří:

a) **testovací plán** – je výchozí dokument pro celé testování. Řídi celý proces testování. Obsahuje definici a rozsah testů, co je předmětem testování, zajištění zdrojů, určení zodpovědnosti, identifikace rizik atp. Vytváří ho člověk v řídící pozici a je z pravidla prvním vytvořeným dokumentem,
b) **testovací případ** – tvoří základní dokument pro testery. Oproti čistě manažersky zaměřeném testovacím plánu je testovací případ podklad pro testování jednoho konkrétního místa v aplikaci při konkrétní situaci,
c) **testovací skript** – vychází z testovacího případu. Skládá se z jednotlivých kroků, které mají být otestovány. Pro každý krok existuje předpokládaný výsledek. Jednotlivé kroky jsou postupně otestovány a tester uvádí jejich výsledek. Pro automatické testování slouží testovací skript jako přepis testovacího případu do kódu,
d) **testovací scénář** – tvoří kombinaci několika testovacích skriptů. Simuluje konkrétní použití aplikace. Pro příklad se testují procesy způsobem, který bude používán u zákazníka. Jsou řazeny dle návaznosti. Výstup jednoho scénáře tvoří vstupní data pro další scénář.

2.4.4 Metriky testování

Stejně tak, jako je testování software jistou kontrolou jeho kvality, tak i samotné testování musí být kontrolováno. Většinou za účelem zjištění a zhodnocení stavu projektu. Informace o probíhajícím testování mohou posloužit např. k posílení testovacího týmu o další testery, nebo je v opačném případě nasměrovat na jinou práci. Pro potřeby hodnocení testů se využívají tři základní metriky založené na:

a) chybách,
b) testech,
c) kódu.

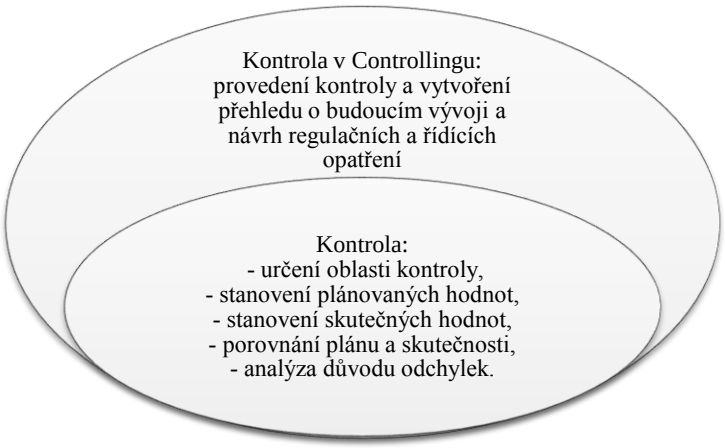
Metriky založené na chybách udávají informace o počtu nalezených chyb v průběhu testování. Samotný celkový údaj o chybách však není směrodatný a nemá velkou vypovídající hodnotu. Proto se u těchto metrik udávají doplňující informace v podobě závažnosti chyby, funkční oblast objevení, času vzniku, typ chyby, stavu opravu chyby atp. Jako příklad uvedu několik užitečných metrik. Metrika **Bug Fix Rate** vyjadřuje procentuální množství již opravených chyb na celkovém počtu chyb nalezených. **Efektivnost testu** udává, kolik procent chyb bylo nalezeno pomocí konkrétního testu. Slouží pro porovnání testů mezi sebou. Testy s malou efektivností mohou být v delším časovém horizontu nahrazeny testy více efektivními. **Rychlost objevení chyby** vyjadřuje počet nalezených chyb za časový úsek. Počet nalezených chyb postupně klesá s blížícím se koncem testování.

Metriky založené na testech počítají s předem známým počtem testovacích scénářů. Předpokládají, že je možné zjistit průběh testování pomocí počtu již provedených testů. Nedostatky této metriky jsou v podobě měnících se náročností jednotlivých testů. Některé testy mají větší váhu.

Metriky založené na programovém kódu vycházejí z počtu procent kódu, který již prošel testováním. Tyto metriky jsou často označovány jako pokrytí kódu (code coverage). [33]

2.5 Controlling

Ve vztahu ke kontrolování samotném se v podnikové praxi a jeho managementu začal uplatňovat pojem controlling. Ačkoliv se u controllingu jedná spíše o kontrolu nákladu. Je možné touto kontrolou měřit kvalitu procesů i v oblasti IT. Controlling je již dnes moderním a zavedeným pojmem. Controlling se stal účinným nástrojem řízení, který podnikům pomáhá zlepšovat procesy např. financování, plánování a rozhodování. Termín controlling se odvíjí od výrazu „control“. V anglickém slovník je definován ve jmenném i slovesném tvaru. Ve jmenném tvaru znamená kontrola, řízení, dohled, vedení a ve slovesném tvaru kontrolovat, řídit, regulovat, ovládat. Controlling úzce souvisí se samotnou kontrolou. Oproti kontrole obsahuje také plánování a řízení.



Obrázek 7. – Kontrola a Controlling
Zdroj: Profesionální controlling – vlastní zpracování [34]

Definice controllingu je v literatuře vykládána odlišně. Některé definice z literatury uvedu.

Podle Eschenbacha „*controlling doplňuje a integruje management jak v koncepčním, funkcionálním a institucionálním smyslu, tak i v personálním smyslu (při vytvoření vlastních controllerů). Controllingová filozofie (software) a infrastruktura controllingu (hardware) jsou sloupy doplňující řízení. S jejich pomocí bude možné dostat pod kontrolu komplexnost řízení podniku.“* [34]

Sekerka označuje controlling za „*nástroj řízení, který má za úkol koordinaci plánování, kontroly a zajištění informační datové základny tak, aby se působilo na zlepšení podnikových výsledků.“*3 [35]

3 Možnosti kontroly vývoje v systémech SAP

3.1 Zákaznický vývoj

Dnešní prostředí IT systémů se zřídka kdy sestává ze stejných řešení. Rozdíly v architekturách IT jsou řešeny nahodile a standardně dodávané business procesy jsou upravovány na základě požadavků zákazníka. Tento proces je nutný, protože standardní procesy nemohou být v souladu se specifickými požadavky zákazníka a většinou není dostupné certifikované řešení. Splynutí standardního SW s úpravami a zákaznickým vývojem se rychle vyvíjí spolu s rychle se měnícími požadavky na business procesy. Nutnost rychlost reagovat na tyto změny sebou většinou přináší implementaci kódu, který neklade důraz na kvalitu, ale spíše na rychlost implementace. Většinou je opomíjeno vypracování důležité dokumentace a analýzy dopadu vývoje na základní business procesy. Změny IT infrastruktury pak s sebou přináší mnoho problémů se znovu implementováním úprav. Události jakou jsou např. update SW, nebo představení nových funkcí, vyžaduje testování zákaznického kódu a úprav, a tím s sebou přináší dodatečné náklady a čas na dobu testování.

Zákaznickým vývojem označujeme úpravy funkcionalit, tvorbu nových reportů, modifikací standardních reportů, propojení s jinými aplikacemi, přizpůsobení standardního SAP modulu či programu potřebám zákazníka. Reporty, funkce, formuláře, webové aplikace, pojmy ve slovníku a zákaznická vylepšení označujeme souhrnně jako objekty.

Zákaznický vývoj probíhá pouze na základě potřeby. Provádí se pomocí programovacích nástrojů. U zákaznického vývoje je možné využívat jak klasické, tak agilní metodiky programování. Agilní programování se prosazuje u menších úprav, kde není IT řízeno projektově. Zákaznický vývoj se liší se dle jazyka a typu vývojového prostředí na:

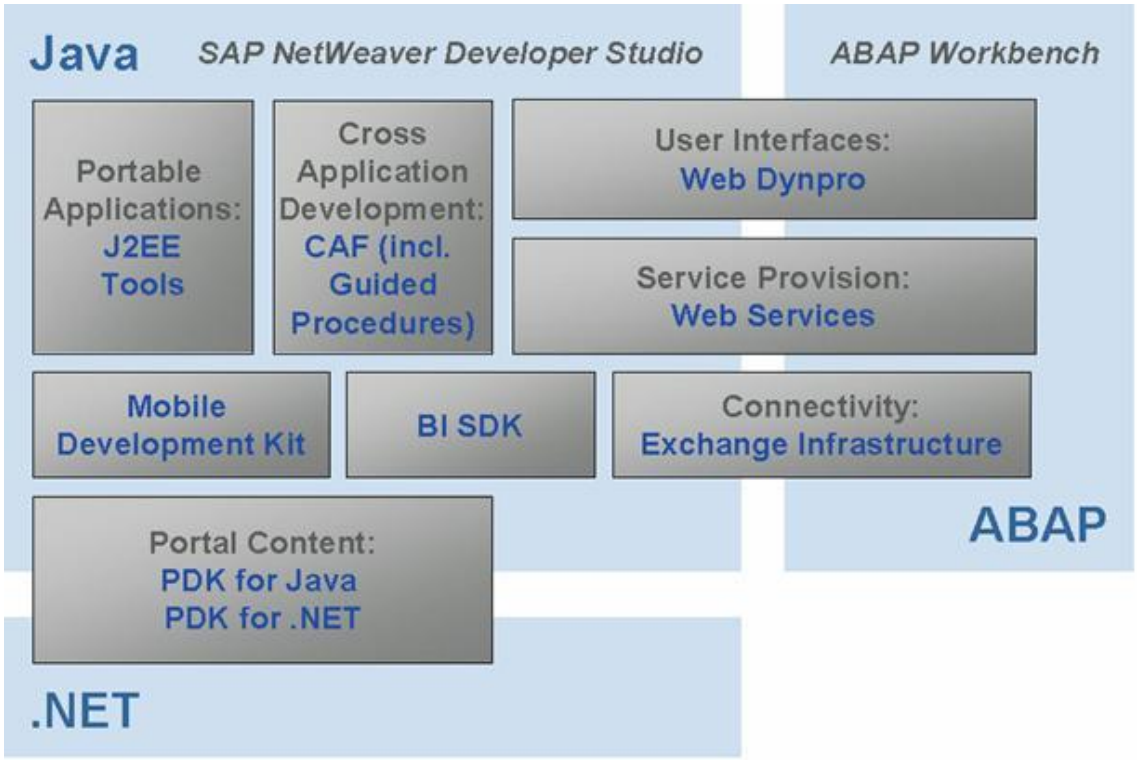
a) jazyk ABAP,
b) jazyk JAVA,
c) SAP Composition Environment.

Jazyk ABAP – zkratka ABAP vznikla ze spojení Advanced Business Application Programming, v originále Allgemeiner Berichts-Aufbereitungs-Prozessor. Základ tomutu jazyku dala v 80. letech 20. století společnost SAP. Na přelomu století bylo provedeno objektové rozšíření jazyka tzv. ABAP Objects. Využívá se společně s jazykem JAVA jako hlavní programovací jazyk pro aplikační server SAP. Většina samotného systému SAP je naprogramována pomocí ABAP. Je součástí komponenty SAP NetWeaver Application Server kde probíhá samotný vývoj v nástroji ABAP Development Workbench.

Jazyk JAVA – Pro zjednodušení vývoje pro informační systém SAP zvolila společnost programovací jazyk JAVA založený na standardu Java EE. Pro vývoj v prostředí SAP slouží SAP NetWeaver Developer Studio. Je založeno na platformě Eclipse, která je open source. Skládá se z několika vrstev, takže vývojář nemusí pro vyvinutí další vrstvy využívat jinou aplikaci.

SAP Composition Environment – jedná se o platformu, která umožňuje tvorbu a provoz aplikací založených na orientaci na služby. Nabízí řadu možností pro integraci nových i stávajících služeb. Slouží pro spojení všech potřebných nástrojů pro tvorbu služeb a uživatelských rozhraní. [38]

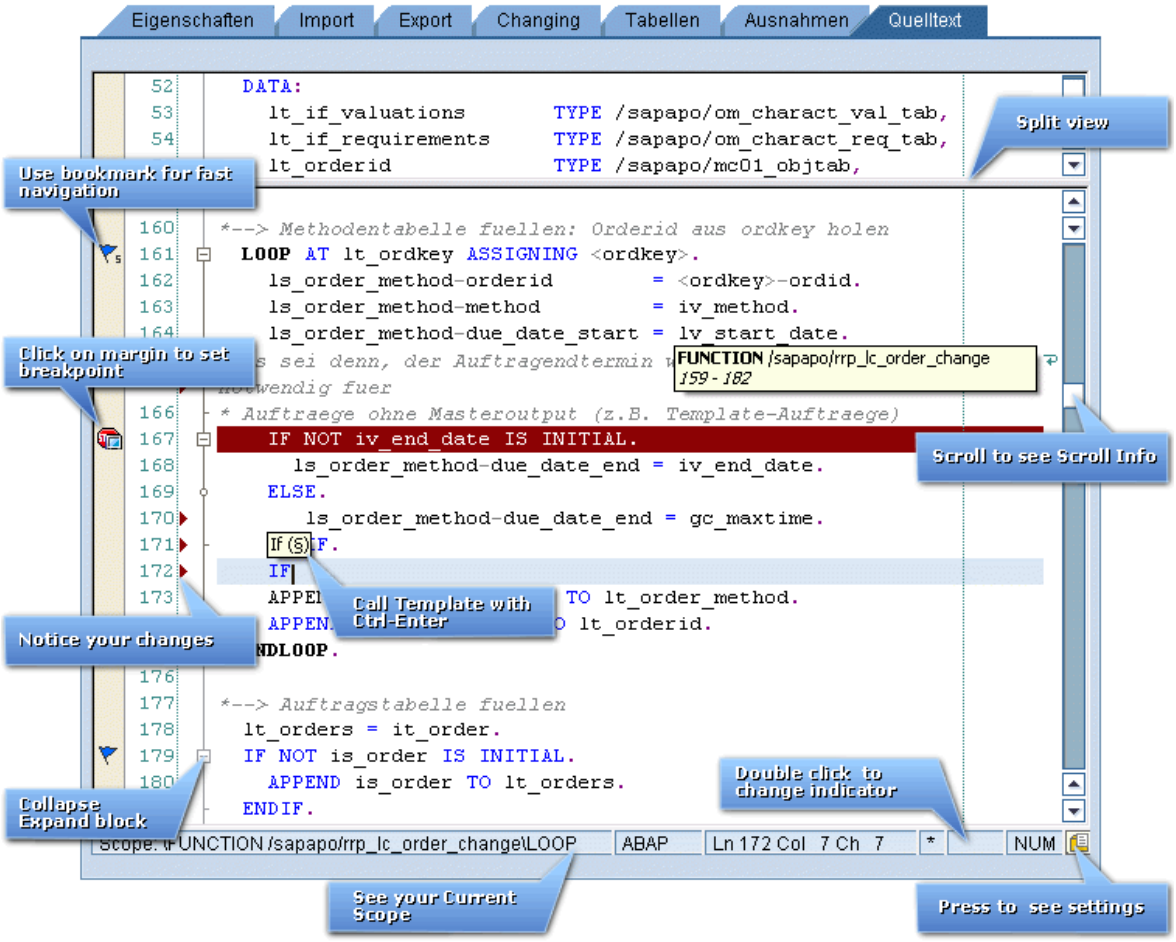
Vše je zastřešeno produktem SAP NetWeaver. Používá se pro zákaznický vývoj, integraci aplikací s ostatními aplikacemi a systémy, tvorbu aplikací pomocí ABAP, C, C++, Java EE, dále spolupracuje s technologiemi Microsoft .NET a IBM WebSphere.



Obrázek 8. – Přehled architektury

Zdroj: help.sap.com

Vývojové prostředí ABAP - Základním nástrojem pro tvorbu nebo úpravy kódu ABAP je ABAP Editor. Jedná se o klasický nástroj pro programování. Uživatelům nabízí užitečné nástroje, které pomáhají v tvorbě kódu. Mezi funkce patří zvýraznění syntaxe, automatické doplňování, předlohy kódu, klávesové zkratky, nápověda, rozšíření schránky atp.



Obrázek 9. – Funkce nového Front-End ABAP Editor

Zdroj: help.sap.com

3.2 Proces zákaznického vývoje ABAP Workbench

Skládá se z několika fází, které na sebe navazují. Pro zahájení další fáze je nutné splnit předem definovaná kritéria. Sap zákaznický vývoj popsal v následujícím procesu.

a) funkční návrh (Functional Design),
b) technický návrh (Technical Design),
c) programování (Code),
d) testování jednotek (Component Testing).

3.2.1 Funkční návrh (Functional Design)

Slouží pro technologicky nezávislé definování požadavků. Definici a komunikaci provádí jednotlivý uživatelé nebo business oblasti. Funkční návrhář (zpravidla zástupce business oblasti v IT, nebo vývojář) přezkoumá a zhodnotí dopad požadavků. Pomocí gap analýzy určí, které kroky musí být provedeny k zajištění požadovaného výsledku. Předtím, než začne vytvářet dokumentaci požadavků, podmínky testování a hlášení chyb, na základě zkušeností a znalostí zhodnotí, jestli není možné využít již dodaných standardních částí systému. Čím více je detailní funkční dokumentace, tím jednoduší je technický návrh a tím méně času je nutné vynaložit na řešení nejasností a dotazů.

Procesní tým zajišťuje požadavky uživatele a převede je do funkční specifikace. Člen procesního týmu odpovídá za zmapování požadavků v systému SAP. Určuje uložení datových prvků (v jaké transakci a obrazovce). Na tomto základě určí, v jaké tabulce a poli se nachází klíčové datové prvky. Tyto informace jsou základem pro tvorbu technické dokumentace. Procesní tým je odpovědný také za tvorbu testovacích scénářů. Tyto scénáře slouží jako podklad pro testování. Vývojový tým slouží jako technický poradce procesnímu týmu. Pomáhá při tvorbě funkční specifikace.

Funkční návrhář zodpovídá za identifikaci možných funkčních chyb. Příklad funkční chyby je např. „typ materiálu nenalezen“. Vzniká dokument „Error-Handling Requirements“, který by měl popisovat následující činnosti:

a) **potencionální chyby** – uvádí chyby, které mohou nastat,
b) **typy chyb** – označuje typ potencionální chyby dle její závažnosti (varování, chyba, kritická chyba),
c) **způsob oznámení** – popisuje činnost oznamování chyb. Definuje formu komunikace a její obsah,
d) **matice oznámení** – určuje, kteří pracovníci budou informování,
e) **formulář oznámení** – určuje detaily a formu ohlášení chyb.

Fáze končí schválením a výstupní kontrolou. Funkční návrh je zkontrolován vedoucím procesního týmu, architektem procesu a vývojovým týmem.

3.2.2 Technický návrh (Technical Design)

Na základě business požadavků uvedených ve funkčním návrhu definuje detailní technické řešení. Technický návrhář pečlivě zkontroluje funkční požadavky a navrhne odpovídající řešení. Snahou je využít již existující řešení. Pokud neexistuje řešení, vytváří návrhář detailní technickou dokumentaci s ohledem na výkon, optimalizaci a testovací případy. V úvahu by měl brát uživatelské rozhraní SAP (doplňky, modifikace) ve smysl použití stejné terminologie. Špatně použitá terminologie může způsobit nejasnosti a následně ztrátu času při jejich řešení. Důležitá je také schopnost zhodnotit jak změny ve funkčním návrhu ovlivní změny v technickém návrhu.

Odpovědností procesního týmu je poskytnout pomoc vývojovému týmu při objasnění požadavků ve funkční specifikaci. Pokud technické řešení neumožnuje zrealizovat veškeré požadavky kvůli nějakým omezením, procesní tým je informován a snaží s vývojovým týmem nalézt kompromisy ve funkčním návrhu.

Technický návrh by měl poskytnout konkrétní a intuitivní vysvětlení funkčních požadavků. Výsledkem každého technického návrhu může být vývoj jednoho nebo více programů. Pro každý program je důležité uvést:

a) **typ programu** – určuje, jestli se jedná o běžný nebo nový program, pro nový program definuje jméno,
b) **SAP aplikace**,
c) **vývoj tříd** – popisuje vývoj tříd použitých v novém programu,
d) **skupina oprávnění** – definuje oprávnění a přidělení rolí,
e) **transakce SAP** – skupina transakcí související s vyvíjeným programem,
f) **vstupní a výstupní soubory**,
g) **vývojový diagram** – při složitém programu je vhodné uvést i vývojový diagram.

Pro fází testování je nutné definovat akceptační kritéria testování. Uvedeny by měly být také očekávané výsledky na základě vstupních a transakčních dat. V technickém návrhu je důležité dodržet standardní vzhled pro každý dokument, přesnost a úplnost obsahu.

3.2.3 Programování (Coding)

Na základě technického návrhu je zahájeno programování. Výsledek by měl odpovídat uživatelským požadavkům bez zjevných chyb. Procesní tým pomáhá vývojovému týmu upřesnit funkční požadavky. Vývojový tým zajišťuje vývoj, kontrolu a opravu chyb. Informuje procesní tým v případě technických omezení.

Z hlediska kontroly programování v jazyce ABAP je potřeba, aby programátor dával pozor na následující:

a) základní zpracování chyb a zpráv,
b) chybějící nebo prázdné vstupní nebo výstupní soubory, adresáře a proměnné,
c) chyby ve formátech výpočtech (dělení nulou, přetékání),
d) formát vstupních a výstupních polí,
e) správnost použití všech funkcí,
f) výpočetní chyby,
g) časová razítka,
h) komentář kódu.

Testovací plán je nutné upravit a doplnit o informace, které se ukázaly být důležité při fázi programování a nejsou již uvedeny. V testovacím plánu jsou vyznačeny důležité kroky, tímto je testovací plán uzavřen a přechází se do fáze testování.

3.2.4 Testování jednotek (Component Testing)

Úkolem testování jednotek (komponent) je zajištění bezproblémové funkčnosti pokud jde o technické specifikace v provozu ve vývojovém prostředí. Test ve vývojovém prostředí provádí sám vývojář. Dokumentace k testování komponent slouží jako základ k budoucím úpravám a jako základ pro regresní testování. Procesní tým pomáhá při testování v podobě konfigurace nebo správy vstupních dat. Vývojový tým otestuje vývoj a zajistí opravu chyb.

Aby byl test považován za kompletní, ve smyslu obsahu testovaných částí, je vhodné testovat každou klíčovou oblast:

a) kontrola podmínek výběru,

b) negativní a pozitivní hodnoty,
c) funkce WHERE,
d) deklarace Read, Extract, Clear, Assignment,
e) kontrola cyklů (nekonečná smyčka),
f) ověření funkcí jednotlivých modulů,
g) uživatelské rozhraní,
h) návrh výstupu,
i) online dokumentace.

Ve fázi testování je možné odhalit nedostatky, které si žádají změnu v technickém nebo funkčním návrhu. Programátor se ve spolupráci s technickým návrhářem a vedoucím vývoje snaží najít vhodné řešení. V případě zásahu do funkčního návrhu se tyto změny projednají na setkání s funkčním návrhářem. [39]

3.3 SAP Solution Manager

SAP Solution Manager (SOLMAN) je v současné době hlavním a základním nástrojem pro správu celého landscape SAP. Je použitelný i pro ostatní systémy. Obsahuje metodiku a nástroje pro řízení celého životního cyklu aplikace. Zajišťuje nástroje pro návrh, implementaci, provoz a optimalizaci. S rostoucím počtem systémů a aplikací, které společnost nabízí svým zákazníkům, se znesnadňuje jejich údržba. SOLMAN se snaží zjednodušit a centralizovat správu těchto systémů. Používá k tomu řadu různých nástrojů:

a) Central System Administration,
b) Project Management,
c) Test Management,
d) System Monitoring,
e) Business Process Monitoring,
f) E2E Root Cause Analysis,
g) IT Technical Reporting,
h) Centralized Alerting,
i) Installation Keys,
j) Early Watch Reporting,

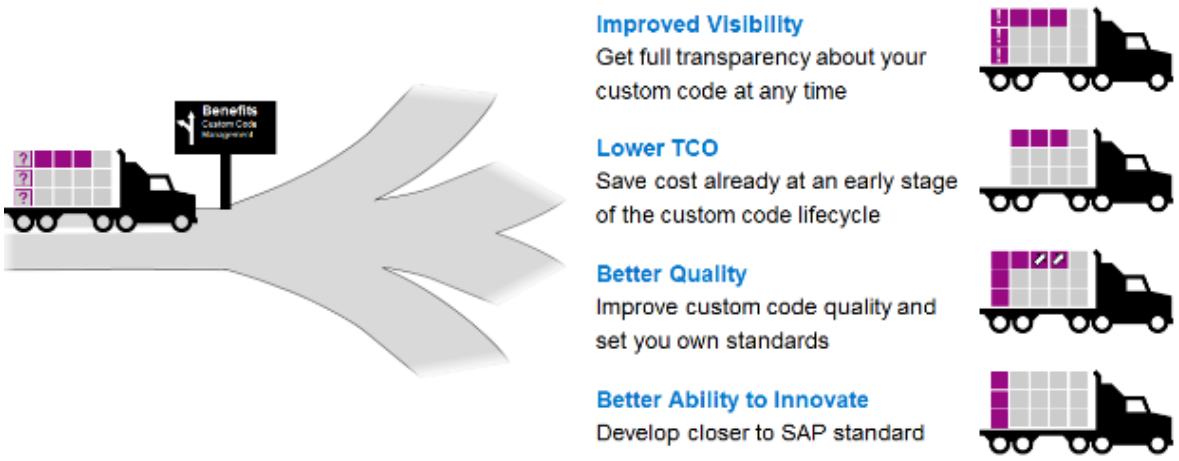
k) Change Management (Change Request Management & Maintenance Optimizer),
l) Service Desk. [17]

SOLMAN podporuje správu a optimalizaci zákaznického vývoje a umožňuje individuální úpravy standardně dodávaných řešení. K těmto úpravám a správě slouží mnoho nástrojů, které jsou obsaženy přímo v SOLMANu v rámci ALM (Application Lifecycle Management). S pomocí těchto nástrojů je možné analyzovat zákaznický vývoj a úpravy v rámci použití v systémech a získat tak celkový přehled zákaznického vývoje. Na základě této analýzy je pak možné lépe kontrolovat vývoj v systémech SAP. Cílem je zlepšit kvalitu vývoje a implementace spolu s omezením počtu a dopadu vývoje na ostatní objekty. Toto opatření vede ke snížení nákladů na vývoj a k lepší správě vývoje celkově.

3.4 Custom Code Management a Custom Code Lifecycle Management

Custom Code Management (CCM) zastřešuje celý proces kontroly a zlepšení zákaznického vývoje pro systémy SAP. S jeho pomocí je možné získat několik výhod:

a) úplný přehled o zákaznickém vývoji,
b) ušetření prostředků v raných fázích vývoje,
c) identifikace a odstranění nepoužívaného kódu,
d) zlepšení kvality zákaznického vývoje,
e) přiblížení standardům.

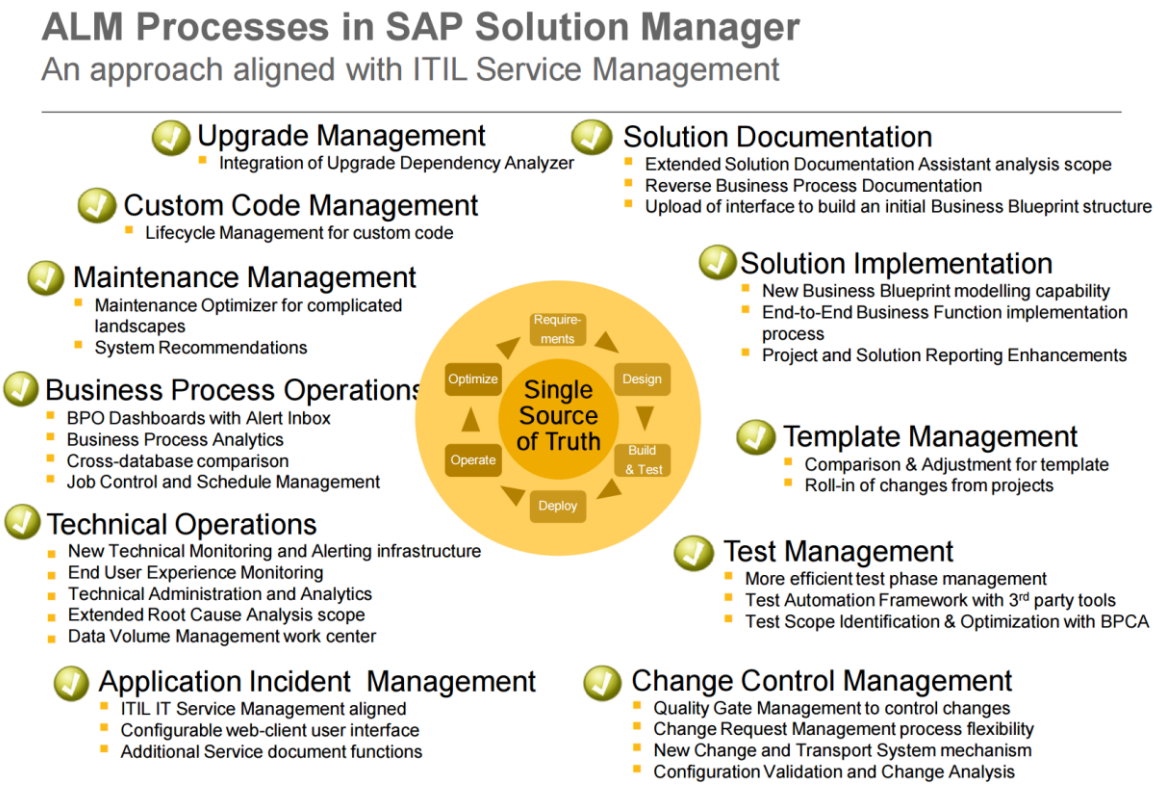


Obrázek 10. – Výhody Custom Code Management

Zdroj: SAP Community Network, http://scn.sap.com/docs/DOC-54888

Custom Code Lifecycle Management (CCLM) je funkce v programu SAP Solution Manager, která slouží pro efektivní správu zákaznického vývoje. Tato funkce byla přidána v poslední verzi 7.1. Spíše než na samotný vývoj ve smyslu programování, se zaměřuje na sledování zákaznického vývoje uvnitř systému. Poskytuje metodiku, která pomáhá zlepšovat kvalitu zákaznického vývoje s pomocí komplexních nástrojů pro monitoring a správu. CCLM doplňuje řadu již existujících nástrojů v SOLMAN a byl vyvinut přímo pro podporu zákaznického vývoje v programovacím jazyce ABAP. Hlavní roli v CCLM pro ABAP zastává Code Inspector. CCLM s pomocí funkce Code Inspector zajišťuje pravidelnou kontrolu objektů během celého jejich životního cyklu.

CCLM je součástí správy celého životního cyklu aplikace Application Life Management (ALM), která nabízí procesy, nástroje, best practices a služby ke správě systémů SAP, včetně systému mimo SAP. Proces kopíruje doporučení ITIL.



Obrázek 11. – ALM Procesy

Zdroj: help.sap.com

CCLM používá různé nástroje na monitoring zákaznického vývoje. Na základě automatického sběru dat vytváří analýzy o využití objektů a jejich porovnání se standardem. Mezi hlavní nástroje patří:

a) **konfigurace –** umožňuje základní úpravu nastavení CCLM. Patří sem nastavení, ze kterého systému se sbírají data a detailní nastavení extraktorů dat.
b) **zobrazení objektů** – zobrazuje výčet všech upravených objektů. Obsahují informace, kdo je autorem objektů, datum přidání, datum úpravy, datum poslední změny.
c) **ad-hoc reporting** - Specifické vyhledávání je možné upravit podle určitých kritérií. [40]

3.4.1 Code Inspector

Zákaznický vývoj je nutnou součástí implementace systémů SAP. Standardní dodávaný software nemůže vyhovět zákazníkovi ve všech směrech. Výsledkem je, že zákazníci upravují systém do požadovaného stavu. Kvalita zákaznického vývoje má velkou váhu na celkový výkon, bezpečnost a použití systému.

Code Inspector je nástroj pro kontrolu kódu ABAP. Kontrolu umožňuje z hlediska správnosti, výkonu, bezpečnosti, spolehlivosti a poskytuje také statistické informace. S pomocí nástroje Code Inspector je možné sledovat všechny důležité aspekty vývoje a porovnávat je se zásadami kvality. Vývojářům s pomocí zpráv a notifikací pomáhá dodržovat vývojové standardy. Upozorňuje na nedostatky v optimalizaci kódu. Kombinací různých typů kontrol je možné kontrolovat specificky definované vlastnosti kódu z různých úhlů pohledu. Kontrola je prováděna na základě předdefinované matice konfliktů, která se dá modifikovat podle interních pravidel vývoje. Mezi jednotlivé kontroly patří:

a) **ABAP Test Cockpit**,
b) **kontrola syntaxe**,
c) **výkon** – kontroluje použití funkcí, které mohou mít negativní vliv na výkon programu. Jedná se zejména o WHERE, SELECT, UPDATE a DELETE,
d) **bezpečnost** – kontrola kritických prvků ohrožujících bezpečnost, např. přístupy do databází jiných klientů atp.,
e) **robustnost**,
f) **programovací konvence**,
g) **jmenná konvence**,
h) **funkce vyhledávání**,
i) **měření a statistiky** – kontrola celého programu. Popisuje využití proměnných. [41]

3.4.2 Custom Development Management Cockpit

S růstem počtu objektů upravených zákazníkem, rostou také nároky na jejich správu. Například není možné jednoznačně říct, jaké objekty jsou důležité a používané, a které

objekty jsou nadbytečné a nepoužívají se. Během upgradů systému, nebo instalace nového servisního balíčků je nutné zkontrolovat funkce všech zákaznických objektů. U každého objektu je důležité zkontrolovat dopad na změněnou konfiguraci systému. Pokud je objekt nadbytečný před změnou systému, jeho kontrola po změně je zbytečná. Nedá se ani určit přesný čas na upgrade či změnu systému s ohledem na zdlouhavou kontrolu zákaznických objektů. K tomuto faktu přispívá nedostatek informací o těchto objektech. Společnosti s centrálním vývojem pro více systémů čelí problému kontroly, jestli jsou objekty aktuální a fungují i v ostatních systémech. Tento problém se společnost SAP snaží vyřešit nástrojem Custom Development Management Cockpit (CDMC). Tento nástroj nabízí spojení business procesů s příslušnými objekty. Upgrade systému ovlivňuje určité business procesy, ty jsou propojeny s příslušnými objekty a tím je určena oblast testování. Odpadá potřeba testovat všechny objekty. Do CDMC patří:

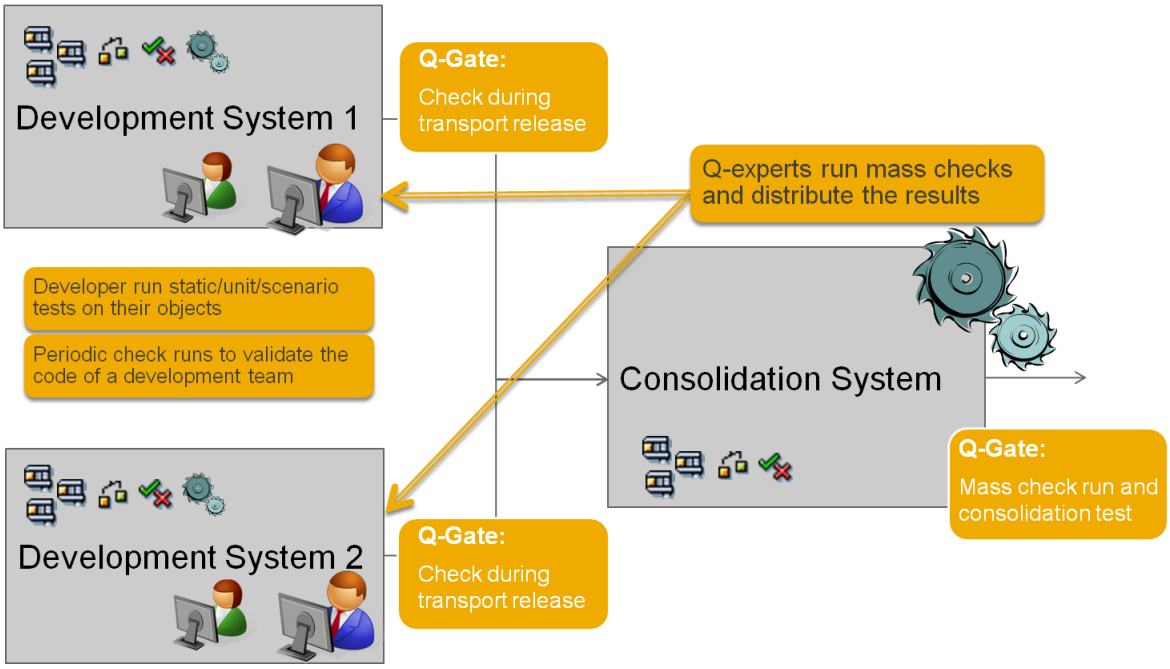
a) **Clearing analýza** – analyzuje upravené objekty a změny od standardních SAP objektů s cílem určit jejich využívání a změny. Nepoužívané objekty je možné vymazat,
b) **Upgrade / Change impaktní analýza** – určuje dopady zákaznického vývoje na změny nebo upgrade systému a potřebný čas pro jejich přizpůsobení,
c) **Change a Transport systém analýza** – analyzuje objekty v transportech. Umožňuje určit četnost použití a porovnává tyto objekty napříč systémy. [42]

3.4.3 ABAP Test Cockpit

Chyba způsobená ve vývoji kódu ABAP, může mít velké důsledky v oblasti kritických business procesů. Pokud kritické činnosti závisí na kvalitě kódu, je přímo nutné kód kontrolovat a zkvalitňovat. Cílem je největší kvalita. Aby SAP pomohl tento cíl splnit, vyvinul nástroj ABAP Test Cockpit (ATC). Jedná se o hlavní nástroj pro dodržování Quality Assurance pro vývoj v ABAP. ATC bylo nejprve vyvinuto pro interní účely programování přímo ve firmě SAP. Po pozitivních ohlasech při zkoušení u několika zákazníků, se firma rozhodla tento nástroj implementovat do procesu kontroly kódu pro zákaznický vývoj.

Hlavní funkcí tohoto nástroje je provádění statických a jednotkových testů programů v ABAP. ATC je založen na programu Code Inspector a je přímo integrován do procesu vývoje. Zákazníci mohou spouštět již nastavené kontroly. ATC zavádí nové procesy pro zajištění kvality, například brány kvality (Quality Gates), robustní schvalovací proces výjimek a pravidelné regresní testy.

Jednou bránou kvality je například automatický test transportu. Program, který nesplní kvalitativní požadavky, bude vyjmut z transportu. Mezi další funkce patří schvalování výjimek (neoprávněně programem nahlášená chyba) s automatickou notifikací odpovědných osob, provádění jednotkových testů, zobrazování statistik dle různých kritérií a další. [44]



Obrázek 12. – ABAP Test Cockpit

Zdroj: ABAP Test Cockpit – an Introduction to SAP's new ABAP Quality Assurance Tool [44]

4 Optimalizace procesu kontroly vývoje v SAP systémech

Tato kapitola popisuje případovou studii přímo u uživatele systému SAP s rozsáhlým zákaznickým vývojem. Kontrola vývoje v podniku nebyla nastavena. S přibývajícím objemem zákaznického vývoje bylo nutné optimalizovat proces kontroly vývoje.

4.1 Doporučení SAP

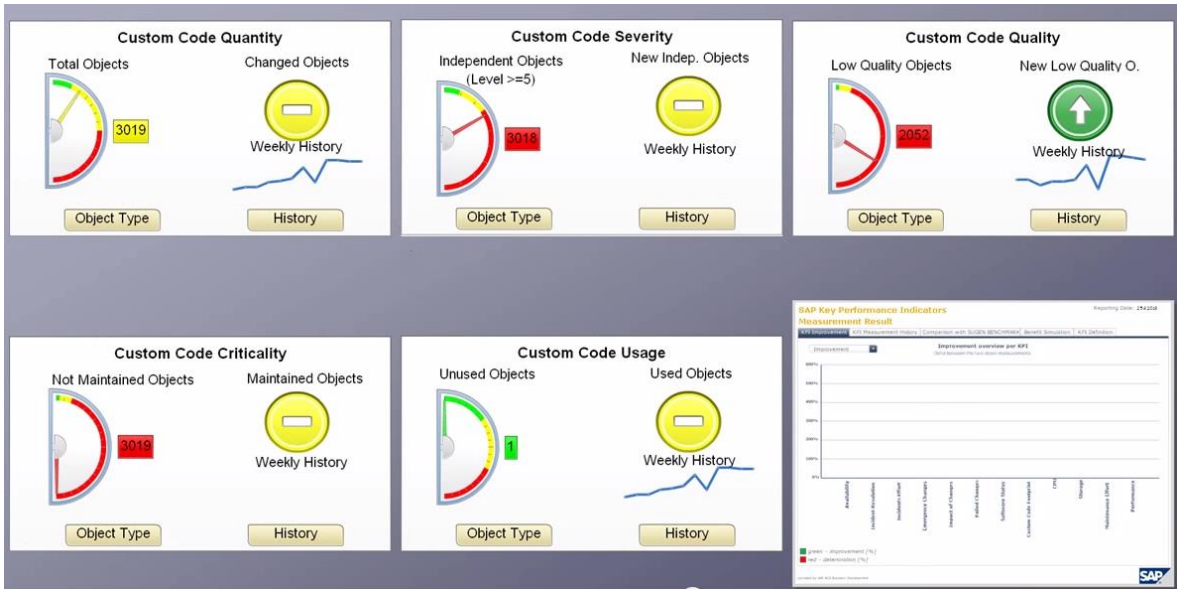
SAP pro optimalizaci kontroly zákaznického vývoje doporučuje ve spojení se SAP Solution Managerem, který nabízí SAP Solution Manager Custom Code Management Blueprint, postupovat na základě čtyř po sobě jdoucích kroků:

a) **Custom Code Transparency** - nejprve je nutné zjistit aktuální stav zákaznického vývoje. K tomu slouží **Custom Code Lifecycle Management (CCLM)** v aplikaci SAP Solution Manager, který obsahuje různé funkce pro detailní analýzu zákaznického vývoje,

b) **Custom Code Control –** druhým krokem je získání kontroly nad vývojem. S pomocí nástrojů obsažených v **Custom Code Analysis** získá zákazník detailní přehled. Jedním nástrojem je například Clone Finder, který slouží pro detekci kopií objektů zákaznického vývoje vytvořených ze standardně dodaných objektů. Ukazuje procento shody a také porovnává shody programů mezi systémy,

c) **Custom Code Optimization –** následující krok je zaměřen na optimalizaci kódu. K tomu je možné využít nástroj **Custom Development Management Cockpit (CDMC**), který provádí analýzy o využití objektů v jednotlivých business procesech. Dále nástroj **ABAP Code Inspector**, pomocí této funkce se testují jednotlivé objekty nebo skupiny objektů. Testuje se výkon, bezpečnost, chybovost atp.,

d) **Custom Code Reporting** - posledním krokem je reporting výsledků. SOLMAN v tomto ohledu nabízí takzvané „dashboards“. Pomáhají ve srozumitelné podobě zhodnotit stav zákaznického vývoje. K tomu jsou používány různé grafy, viz obrázek 11. Dashboards jsou standardně přednastaveny, ale je možné je libovolně upravovat a definovat vlastní náhledy. Ukazují například množství zákaznického kódu, optimalizaci kvality vývoje na základě úspěšnosti použití, množství nepoužívaného kódu a podobně. [43]

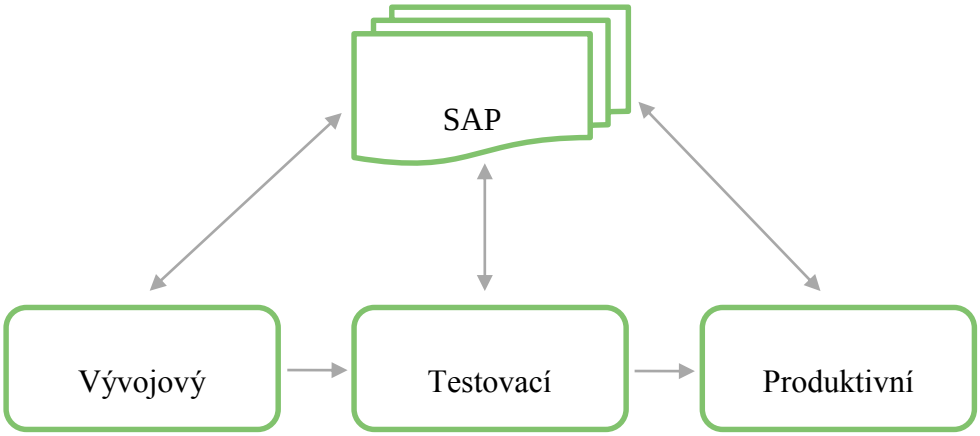


Obrázek 13. – SOLMAN Dashboards

Zdroj: printscreen SAP Solution Manager – vlastní zpracování

4.2 Výchozí stav

Podnik využívá rozdělení systému do tří nezávislých instalací. Jak je uvedeno v kapitole 1.9, toto rozvržení slouží k minimalizaci rizika nechtěného zásahu do systému. Uživatelé pracují pouze s produktivním systémem, vývoj probíhá na vývojovém systému a testování probíhá na testovacím systému. Testovací systém používá kopii skutečných dat z produktivního systému. Tím je zajištěno otestování reálného chování programu. Program, který plní požadavky, je žadatelem otestován v testovacím systému. Po úspěšném otestování je proveden transport vyvinutého objektu do produktivního systému.



Obrázek 14. – Instalace systému SAP v podniku
Zdroj: vlastní zpracování

Samotný postup úprav a tvorby kódu je popsán v příručkách pro vývojáře. Tyto příručky popisují pouze jak postupovat při vývoji programu např. jmenná konvence, uložení do transportu atp.

Hlavním nedostatkem aktuálního procesu vývoje je skutečnost, že není popsána kontrola kvality a veškerou odpovědnost za kvalitu a bezchybnost kódu přebírá vývojář. Kód již dále není kontrolován další osobou nebo automatickým systém pro kontrolu kvality a identifikaci rizik. Bez fungující kontroly kvality není možné odhalit prvky kódu, které můžou být chybové, rizikové z hlediska bezpečnosti, nadměrně zatěžující systém vinou špatně odladěné funkce atp. Do bezpečnostního rizika se řadí únik citlivých dat podniku, který může zapříčinit velké škody. Nebezpečí úniku dat přichází z větší míry zevnitř podniku. Může ho zapříčinit i nechtěná chyba v aplikaci. Kontrola zákaznického vývoje si klade za cíl toto nebezpečí eliminovat.

Výsledný program prochází testováním pouze na základě funkčnosti v podobě funkčních, integračních, systémových a akceptačních testů. Testy jsou omezeny jen na nezbytný objem. Jednotlivé fáze vývoje a konfigurace v systému SAP ERP z částí koresponduje s životním cyklem vývoje software.

a) **analýza a implementace** – se provádí na základě podnětu klíčového uživatele odpovědného za jednotlivý business úsek. Klíčový uživatele provede analýzu

požadavků na systém a zpracuje průvodní dokument k provedení úpravy. Úpravy se dělí do dvou kategorií customizing a workbench. Customizing pracuje se standardními prvky systému, které jsou následně upraveny dle požadavků podniku. V případě workbench se jedná o přidávání nových prvků do systému SAP pomocí programového prostředí. Implementace je prováděna systémovým organizátorem,

b) **transport do testovacího systému** – po vyvinutí, je funkcionalita aplikována na určený testovací systém za použití transportní cesty,
c) **testování** – je prováděno klíčovým uživatelem,
d) **transport do produktivního systému** – nastává po úspěšném otestování.

4.2.1 Role pro kvalitu vývoje zákaznických programů

Nejprve bylo nutné definovat základní procesní role pro zákaznický vývoj. Každá role má přidělené činnosti, které vykonává a za které nese odpovědnost. Role korespondují s doporučením SAP, ale jsou upravené pro konkrétní potřeby podniku.

Administrátor:

a) přiděluje a spravuje oprávnění jednotlivým uživatelům v procesu kvality,
b) spravuje design Team WEB a aktivuje nepravidelně doplňované části WEB (linky, kontakty, maily),
c) generuje nepravidelně se opakující akce v systému (prvotní nastavení statistik, aktualizace team web stránek, kontrola správnosti navrhovaných řešení pro odstranění chyb),
d) spravuje obsah nápovědy v systému.

Auditor:

a) provádí pravidelně kontrolu programů a transportů v systému. S nálezy chodí za programátory a dojednává s nimi postup řešení. Případně eskaluje na koordinátory,
b) informuje pravidelně o výsledcích kontrol programů management a koordinátory modulů,
c) informuje pravidelně o výsledcích kontrol transportů management a HPSM,
d) provádí také kontrolu vyřešení nálezů,

e) v případě projektů se podílí na stanovené strategie nastavení pravidel kvality vývoje.

Management:

a) určuje strategii oprávnění pro jednotlivé uživatele v procesu kvality,
b) sleduje výsledky kontrol programů z hlediska systému a přijímá opatření pro nápravu,
c) sleduje eskalované případy a přijímá opatření pro nápravu,
d) schvaluje řešení pro odstranění chyb.

Koordinátor:

a) schvaluje oprávnění pro jednotlivé uživatele v procesu kvality (vývojáře a externí konzultanty),
b) sleduje výsledky kontrol programů z hlediska svého modulu a vývojářů,
c) projednává nalezené chyby s vývojáři a schvaluje postup pro odstranění chyb, pokud to situace vyžaduje,
d) informuje o chválených postupech auditora.

Development Quality Control Boardu (DQCB):

a) sleduje výsledky kontrol programů z hlediska chyb,
b) navrhuje systémové řešení pro odstranění chyb.

Vývojář:

a) provádí vždy kontrolu svého nového případně starého vývoje podle přijatých opatření (sám si spouští Code Inspector),
b) sleduje pravidelně schválená řešení pro odstranění chyb a provádí nápravu zjištěných chyb,
c) konzultuje s koordinátorem a auditorem aktuální problémy spojené s vývojem,
d) navrhuje opatření a témata k řešení pro radu chytrých.

Organizátor (sekretář) v procesu kvality:

a) generuje pravidelně reporty pro management, koordinátory a HPSM,

b) doplňuje pravidelně Team web (kalendáře, statistiky, grafy, zápisy,
c) organizuje setkání rady chytrých a pořizuje zápisy,
d) organizuje setkání vývojářů,
e) organizuje školení pro vývojáře a začátečníky.

4.3 Navržené řešení

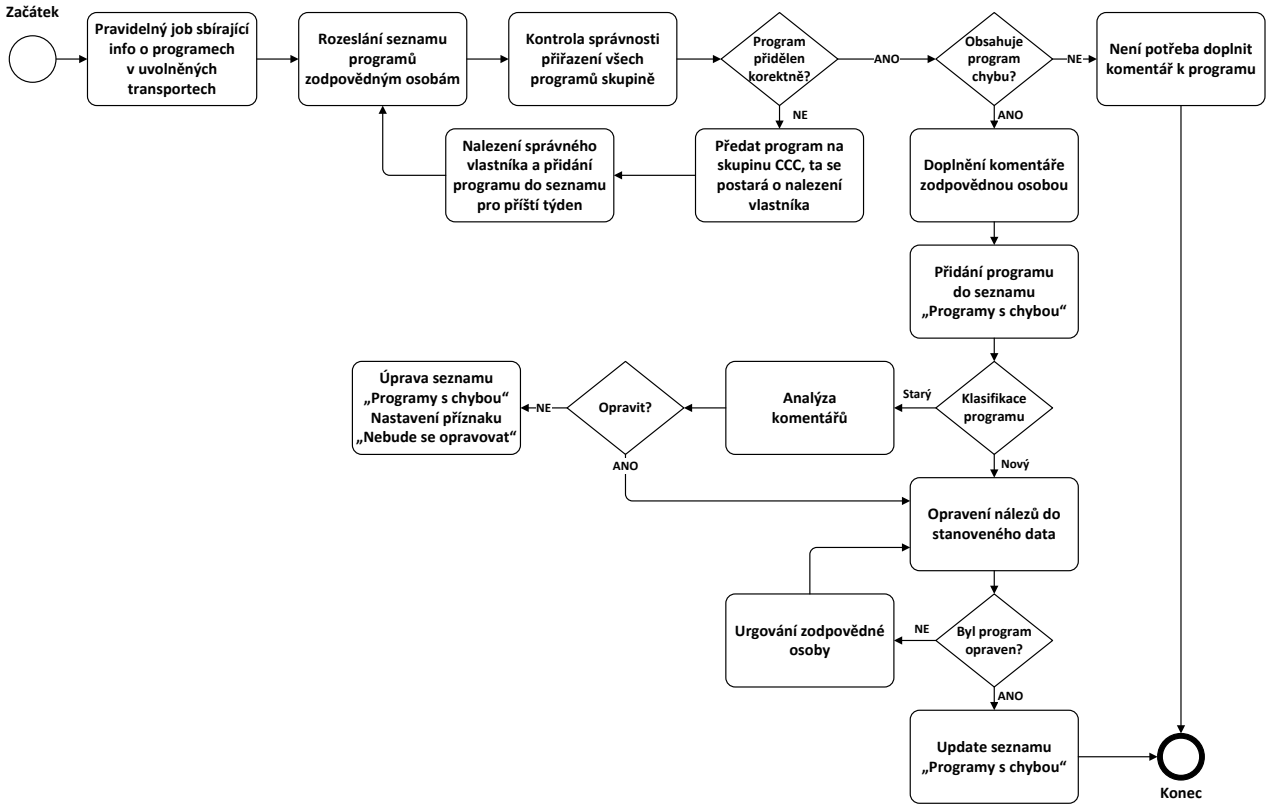
Cílem řešení je optimalizace procesu kontroly kvality software. Produktem optimalizace je příručka SAP Quality Assurance, která stanovuje závazná pravidla a postupy při vývoji nových a při korektuře starých programů v systémech SAP. Řešení bylo optimalizováno pro konkrétní podnik. Z částí koresponduje s doporučením SAP, ale zaměřuje se na kontrolu samotného kódu. Standardní řešení SAP rozšiřuje o kontrolu starších programů dle různých kritérií a dále o možnost uvolňovat programy obsahující chybu, pokud je její výskyt náležitě odůvodněn. Pro analýzu zákaznických objektů bude sloužit proces CCLM v programu SAP Solution Manager. Analýza přinese seznam všech objektů, jejich využití a další důležité informace.

Jako řešení kontroly kódu bylo použití automatického testování pomocí programu Code Inspector. Výsledkem je kontrola všech programů, které se nacházejí v uvolněném transportu. Kontrola probíhá každý den a probíhá na základě naplánované události spouštěné ze systému, na kterém je provozován SAP Solution Manager. Pokud je některý z kontrolovaných vývojových systémů nedostupný, je průběh akce zastaven. Po obnovení dostupnosti systému je provedena kontrola programů od data posledního dne dostupnosti systému.

Seznam s programy obsahující chybu bude zasílán pravidelně každý týden odpovědným osobám (koordinátor či jeho zástupce). Dále bude zasílán měsíční přehled uvolněných programů, u kterých bude požadováno vyjádření, že se jedná nebo nejedná o programy, které spadají do kompetence dané zodpovědné osoby. Po obdržení emailu je týdenní lhůta na odpověď, aby byly okomentovány chybné programy (odůvodnění chyby v programu či vyjádření se, do kdy bude chyba odstraněna) a potvrzena či vyvrácena správnost přidělení programů.

4.3.1 Proces kontroly vývoje

Pro kontrolu zákaznického kódu slouží tento proces:



Obrázek 15. – Proces kontroly vývoje
Zdroj: vlastní zpracování

Tento proces se mimo sběru informací a kontroly zaměřuje na správné zařazení programu odpovědným skupinám programátorů. Po správném zařazení programu je program zkontrolován. Pokud program obsahuje chybu, je nutné chybu okomentovat (může se jednat o přípustnou chybu nebo falešný nález chyby). Pokud se shledá, že je nutné chybu opravit, je tak učiněno co nejrychleji. Chybové hlášení určené jako falešné nebo přípustné se zanese do specifikace kontroly a nebude již znovu označeno za chybu.

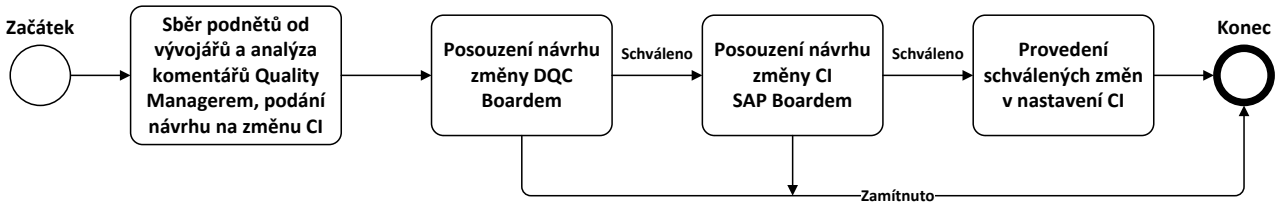
Proces začíná u admina spuštěním pravidelné akce sbírající informace o programech. Následně auditor rozesílá seznam programů odpovědným osobám. V tomto případě jsou zodpovědnými osobami vývojáři. Sekretář poté zkontroluje správnost přidělení programů odpovídající skupině. Pokud je skupina označena nesprávně, spolu s vývojářem případně se skupinou odpovídající za vývojáře, opraví přidělení. Pokud byl program označen jako

program obsahující chybu, vývojář chybu okomentuje. Auditor přidá program do seznamu „Programy s chybou“ a klasifikuje skupinu programu. Po analýze komentáře auditor rozhoduje, jestli je chybu nutné opravit nebo bude chyba zařazena do výjimek. Opravu chyb provádí vývojář. Kontrola provedení oprav včetně urgovaní náleží auditorovi. O stavu celého procesu informuje sekretář pomocí nástrojů reportingu.

Proces kontroly vývoje kontroluje programy již obsažené v produktivním systému. Pro omezení vzniku chyb před transportem programu produktivního systému bylo zavedeno následující opatření. Před uvolněním úlohy vývojářem je nutno zajistit potřebné kroky pro zajištění kvality zákaznického vývoje. Pokud transport obsahuje zákaznický vývoj, pak musí být vývojářem:

a) provedena kontrola pomocí nástroje Code Inspector,
b) uložen jeho výstup (a případně poslán vysvětlující mail),
c) doplněny komentáře a základní údaje o programu.

Návrhy na úpravu nastavení Code Inspector, výsledky kontrol a připomínky dává vývojář. Auditor podněty sbírá a hodnotí. Kontrola podnětů a výsledků kontrol jsou probírány na pravidelných termínech zasedání Development Quality Control Boardu (DQCB). DQCB se skládá z vybraných koordinátorů a zkušených programátorů. Diskutované změny budou předány ke schválení SAP DQCB. Pokud budou změny schváleny, budou uvedeny v platnost.



Obrázek 16. – Sběr podnětů pro zlepšení kvality vývoje
Zdroj: vlastní zpracování

4.3.2 Doplňující informace

Proces kontroly vývoje očekává v pozdější fázi testování v testovacím systému na straně business. Probíhá test funkcionalit a testuje se také nežádoucí chování programu. Programy byly rozděleny na dvě základní skupiny dle stáří programu. Masivní historický vývoj v systému nedovoluje akci uspořádat do jediného kroku. První kategorií jsou tzv. „**staré programy**“, které mají datum vzniku před nasazením kontroly. Programy v této skupině podléhají volnějším pravidlům. Pokud je program v této skupině měněn, tak nesmí obsahovat security chybu, avšak pokud bude program finančně vysoce nákladný na opravu, či při jeho nefunkčnosti hrozí společnosti vysoké finanční ztráty (např. zastavení výroby), bude po domluvě možné, že daný program nebude nutné opravovat. Druhou skupinou jsou „**nové programy**“. Programy v této skupině by neměly obsahovat žádné nálezy chyb. Případně může obsahovat informační hlášení.

Kritické příkazy způsobující v programu Code Inspector hlášení chyby klasifikace security:

a) CALL EDITOR,
b) Dynamic programming with GENERATE &1,
c) WRITE / INSERT / DELETE REPORT / TEXT POOL,
d) WRITE / READ SCREEN,
e) WRITE NAMETAB.

Pokud je nevyhnutelné kritické příkazy použít, je nutné odůvodnit pomocí komentáře při kontrole programu, proč program daný příkaz obsahuje a zda je jeho použití nevyhnutelné.

4.3.3 Doporučení

Pro další zlepšení kvality procesu zákaznického vývoje bych doporučoval více využívat standardně dodávané nástroje pro analýzu (CCLM, CDMC) a reporting (Dashboards). Celý proces vývoje by se měl snažit přiblížit doporučením SAP pro ABAP Workbench popsané v kapitole 3.2:

a) funkční návrh (Functional Design),

b) technický návrh (Technical Design),
c) programování (Code),
d) testování jednotek (Component Testing).

Pro sledování a následnou optimalizaci je nutné zajistit sledování vývoje dle definovaných metrik (KPI). Je možné využít standardních metrik přednastavených v Dashboards. Tyto metriky jsou pouze základní a nemají pro podnik velkou vypovídací hodnotu. Bylo by vhodné definovat vlastní KPI. Například:

a) poměr transportů obsahujících chybu oproti transportům bez chyby,
b) poměr security chyb na celkovém počtu chyb,
c) počet varovných hlášení,
d) počet programů hlásících chybu v produktivním systému.

SAP pro celý životní cyklus aplikace definuje standardní KPI. Tyto metriky je možné sledovat v rámci programu SOLMAN. Doporučení nastavení KPI od SAP pro celý systém:

Score Zones™	Business Continuity	Time to Market	Efficiency	Compliance
Mission	Minimize business impact & Optimize costs	Deliver faster	Achieve more with the same staff	Improve audit ratings
Metrics	▪ Time without outage ▪ System performance ▪ Backlog and throughput ▪ Database growth ▪ No. of redundant custom code objects ▪ No. of open ITSM requests ▪ No. of status iterations in ITSM request processing ▪ SLA fulfillment ▪ Time spent on ITSM request processing ▪ No. of emergency changes	▪ No. of open test cases ▪ No. of open ITSM requests ▪ No. of status iterations in ITSM request processing ▪ SLA fulfillment ▪ Time spent on ITSM request processing ▪ No. of redundant custom code objects	▪ No. of open test cases ▪ No. of open ITSM requests ▪ No. of status iterations in ITSM request processing ▪ SLA fulfillment ▪ Time spent on ITSM request processing ▪ Incident effort ▪ Maintenance effort ▪ No. of redundant custom code objects	▪ Non-compliant user authorizations ▪ Software status ▪ No. of open ITSM requests ▪ No. of open test cases ▪ SLA fulfillment ▪ No. of emergency changes ▪ No. of failed changes

Obrázek 17. – KPI pro Aplication Lifecycle Management
Zdroj: příručka SOLMAN 7.1.Highlights Adoption Approach, help.sap.com

5 Zhodnocení řešení

Proces kontroly zákaznického vývoje se v podniku potýkal s nedostatky, které v dlouhém období představovaly potencionální riziko. Proces bez zajištění odpovídající kontroly kvality ohrožoval kritické business činnosti podniku. Výpadek běžných činností podniku představuje vysoké náklady. Náklady nepředstavuje pouze pozastavení činností běžných uživatelů a důsledky zdržení práce, ale také identifikace a oprava chyby. Pokud je chyba objevena již v rané fázi vývoje, nemusí k těmto situacím vůbec docházet.

Na základě doporučení řešení od společnosti SAP a za pomocí použití nástrojů na správu zákaznického vývoje byl optimalizován proces na kontrolu vývoje. Proces je popsán v příručce na zajištění kvality. Jako řešení byl nasazen proces kontroly vývoje s pomocí nástroje Code Inspector. Kontrola je prováděna na základě předdefinované matice konfliktů modifikovatelné podle interních pravidel vývoje. Probíhá automaticky pomocí jobů.

Před samotným transportem je vývojář povinen dodržovat stanovený postup. Tímto řešením je zajištěna dostatečná kontrola programů před jejich transportem do produktivního systému. Programy jsou také kontrolovány přímo v systému. Zjištěné chyby jsou opravovány a jsou přijímána opatření, aby se již neopakovaly.

Upravením standardního řešení doporučeného společností SAP, je docíleno zlepšení kvality procesu vývoje s ponecháním si možnosti rychlého zásahu do systému. Investováním malého objemu peněžních prostředků na kontrolu zákaznického vývoje, vytváří velké úspory z hlediska ochrany dat.

Závěr

Tato práce představuje možnosti řešení kontroly vývoje v informačních systémech a její optimalizaci, zvláště se soustředí na kontrolu zákaznického vývoje v informačních systémech SAP. V praktické části ukazuje nedostatky v procesu kontroly zákaznického vývoje v podniku. Řešení se snaží tyto nedostatky odstranit a navrhuje další postup.

První kapitola popisuje základní pojmy, se kterými se v práci pracuje. Uvádí specifika vývoje software a životní cyklus vývoje software. Představeny jsou metodiky vývoje software. Na jednotlivých metodikách je znázorněn proces kontroly vývoje. Dále je popsán podnikový informační systém SAP ERP. Práce uvádí základní informace o systému, práce s ním, jeho uživatelích a základním rozdělením systému do modulů.

Druhá kapitola seznamuje s problematikou kontroly software a souvisejících činností. Definuje pojem controllingu, reportingu a důležitost měření kontrolovaných činností. Zabývá se také problematikou testování.

Třetí seznamuje čtenáře s pojmem a potřebou zákaznického vývoje. Detailně popisuje proces zákaznického vývoje. Dále představuje základní nástroj na správu systémů v SAP landscape SAP Solution Manager. Na tento nástroj navazují funkce Custom Code Lifecycle Management, která poskytuje metodiku, která pomáhá zlepšovat kvalitu zákaznického vývoje s pomocí komplexních nástrojů pro monitoring a správu.

Poslední kapitoly popisují případovou studii přímo v podniku využívajícím systému SAP s rozsáhlým zákaznickým vývojem. Nejprve je popsáno doporučení společnosti SAP a následně je uvedeno navržené řešení v podobě procesu kontroly zákaznického vývoje uvnitř podniku. V závěru jsou zmíněna doporučení následného postupu a zhodnocení řešení. Hlavním přínosu práce je soustředění teorie a doporučení v oblasti kontroly vývoje software a následně jejich aplikace. Věřím, že tato práce je vhodným materiálem k pochopení problematiky kontroly vývoje software v informačních systémech SAP.

Seznam použité literatury

Citace

[1] SODOMKA, P.; KLČKOVÁ, H. *Informační systémy v podnikové praxi.* 1. vyd. Praha: Computer Press, 2011. 504 s. ISBN 978-80-251-2878-7

[2] ŠMÍD, V. *Pojem informačního systému*. Brno: Mendelova univerzita v Brně. [online]. [cit. 2015-02-13]. Dostupné online z: http://www.fi.muni.cz/~smid/mis-infsys.htm

[3] ČSN EN ISO 9000:2006. *Systémy managementu jakosti: směrnice pro zlepšování výkonnosti*. Praha: Český normalizační institut, 2001.

[4] ŘEPA, V. *Procesně řízená organizace*. Praha: Grada Publishing a.s., 2012. 304 s. ISBN 978-80-247-7866-2

[5] FOLTÁNEK, V. ING. *Řízení procesů* [online]. Brno: Mendelova univerzita v Brně. [cit. 2015-02-15]. Dostupné online z: https://is.mendelu.cz/dok_server/slozka.pl?id=74928;download=124493

[6] HANDFIELD, R. Ph.D. *What is Supply Chain Management?*. [online]. [cit. 2014-11-28]. Dostupné online z: http://scm.nscsu.edu/scm-articles/article/what-is-supply-chain-management

[7] GIRDHAR, J. *Management Information Systems*. 1. vyd. Oxford University Press, 2013. 640 s. ISBN 978-0198080992

[8] KOLÁŘ, P. *Operační systémy* [online]. Liberec: Technická univerzita v Liberci, 2005-02-01, [vid. 2014-11-07]. Dostupné z: http://www.kai.vslib.cz/˜kolar/os/

[9] VONDRÁK, I. *Úvod do softwarového inženýrství*. Ostrava: Technická univerzita Ostrava. [online]. [cit. 2014-11-11]. Dostupné online z: http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf

[10] MALL, R. *Fundamentals of Software Engineering*. 3. Vyd. Prentice-Hall of India Pvt.Ltd, 2013. 465 s. ISBN: 978-8120338197

[11] *Životní cyklus informačního systému* [online]. [vid. 2014-12-02]. Dostupné online z: http://www.fi.muni.cz/~smid/mis-zivcyk.htm

[12] BECK, KENT a *Manifesto for Agile Software Development* [online]. [cit. 2015-02-03]. Dostupné z: http://agilemanifesto.org

[13] BOEHM, Barry W. *A Spiral Model of Software Development and Enhancement*. TRW Defense Syst. Group, Redondo Beach, CA, roč. 21, č. 5, 1988. ISSN: 0018-916

[14] BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů*. 1. vyd. Praha: Grada, 2005. 163 s. Management v informační společnosti. ISBN 80-247-1075-7

[15] ČSN EN 60300-3. *Management spolehlivosti*. 2009

[16] YANG, G. *Life cycle reliability engineering. John Wiley and Sons*. 2007. 517 s. ISBN 0471715298

[17] KŘIVSKÝ, P. *Řízení změn v informačních systémech SAP*. Liberec, 2012. Bakalářská práce. Technická univerzita v Liberci, Ekonomická fakulta.

[18] *Scrum Alliance Core Scrum* [online]. [cit. 2015-02-10]. Dostupné z: https://www.scrumalliance.org/why-scrum/core-scrum-values-roles

[19] KENT, B. *Extreme Programming Explained: Embrace Change*. 2. vyd. Addison-Wesley, 2004. 224 s. ISBN: 978-0321278654

[20] KADLEC, V. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press, 1. vyd. 2004. 278 s. ISBN 80-251-0342-0

[21] SUTHERLAND, K.; Schwaber, J. *Scrum guide*. [online]. [cit. 2015-03-15]. Dostupné z: http://www.scrumguides.org

[22] FIBÍROVÁ, J. *Reporting: moderní metoda hodnocení výkonnosti uvnitř firmy*. 3. vyd. Praha: Grada, 2010. 224 s. ISBN 978-80-247-2759-2

[23] MACHAČ, O. *Reporting jako součást informačního systému podniku*. [online]. [cit. 2015-01-23]. Dostupné z: http://www.systemonline.cz/clanky/reporting.htm

[24] PROCHÁZKA, J.. *Softwarové inženýrství*. Ostrava: Ostravská univerzita v Ostravě. 2. vyd. 2009. 132 s.

[25] PARMENTER, D. *Key Performance Indicators: Developing, Implementing,and Using Winning KPIs*. John Wiley & Sons. 2011. 256 s. ISBN 978-11-180-4491-9

[26] FLEMMING, A. *Clarification of the defeinitions of SQA and SQC*. [online]. [cit. 2015-03-15]. Dostupné z: http://www.sqa.net/softwarequalitycontrol.html

[27] GRADY, B. *Software Metrics: Establishing a Company-Wide Program*. Prentice Hall, 1987. 275 s. ISBN: 978-0138218447

[28] CMMI Product Team. *CMMI for Development*, Version 1.3. [online]. [cit. 2015-02-16]. Dostupné z: http://cmmiinstitute.com/resources/cmmi-development-version-13

[29] BUCHALCEVOVÁ, A.; KUČERA, J. *Hodnocení metodik vývoje informačních systémů z pohledu testování*. Systémová integrace, 2008, roč. 15, č. 2, s. 42–54. ISSN 1210-9479

[30] KANER, C. *Testing Computer Software*, 2. vyd. Wiley, 1999. 480 s. ISBN: 978-0471358460

[31] ROUDENSKÝ, P.; HAVLÍČKOVÁ, A. *Řízení kvality softwaru*. 1. vyd. Praha: Computer Press, 2013. 208 s. ISBN: 978-80-251-3816-8

[32] ORGENSEN, Paul. *Software testing*. Boca Raton: Auerbach. 2008. 416 s. ISBN 0-8493-7475-8

[33] HUTCHESON, W. *Software Testing Fundamentals*: Methods and Metrics. Wiley Pub. 2003. 408 s. ISBN 047143020X

[34] ESCHENBACH, Rolf a SILLER, Helmut. *Profesionální controlling: koncepce a nástroje*. 2. vyd. Praha: Wolters Kluwer Česká republika, 2012. xiv, 381 s. ISBN 978-80-7357-918-0

[35] SEKERKA, B.; HELMANOVÁ, J.; VACEK, V. *Úvod do controllingu*. Praha: Soukromá vysoká škola ekonomických studií, 2007. 84 s. ISBN 978-80-86744-60-5.

[36] *Clarus Concept of Operations*. [online]. Publication No. FHWA-JPO-05-072, Federal Highway Administration (FHWA), 2005 [cit. 2015-03-10]. Dostupné z: http://ntl.bts.gov/lib/jpodocs/repts_te/14158.htm

[37] 829-2008 IEEE *Standard for Software and Systém Test Documentation*. 2008

[38] *SAP Composition Environemnt* [online]. [cit. 2015-03-01]. Dostupné z: https://help.sap.com/nwce

[39] GHOSE, A. *SAP Custom Development Process*. 2010. [online].[cit. 2015-02-03]. Dostupné z: http://wiki.scn.sap.com/wiki/display/ABAP/ABAP+Development

[40] SAP, *Custom Development Lifecycle Management*. 2015. [online]. [cit. 2015-03-14] Dostupné z: http://help.sap.com/saphelp_sm71_sp10/helpdata/en/2a/544b8edfbd459f97f22e039ba166d6/content.htm

[41] EILENBERGER, R. *Code Inspector*. 2014. [online]. [cit. 2015-03-15]. Dostupné z: *http*://wiki.scn.sap.com/wiki/display/ABAP/Code+Inspector

[42] SAP, Custom Development Management Cockpit. 2015. [online]. [cit. 2015-03-17] Dostupné z: http://help.sap.com/saphelp_sm71_sp10/helpdata/en/d3/4b21cf15464589b84bc63f0da54530/content.htm

[43] TEMPLETON, A.; THOMASIS, T. *Managing Custom Code in SAP*. SAP PRESSS. 2012. 351 s. ISBN 978-15-922-9436-7

[44] KAESTNER, Ch. *ABAP Test Cockpit – an Introduction to SAP's new ABAP Quality Assurance Tool* [online]. [cit. 2015-03-18] Dostupné online z: http://scn.sap.com/docs/DOC-31773

Bibliografie

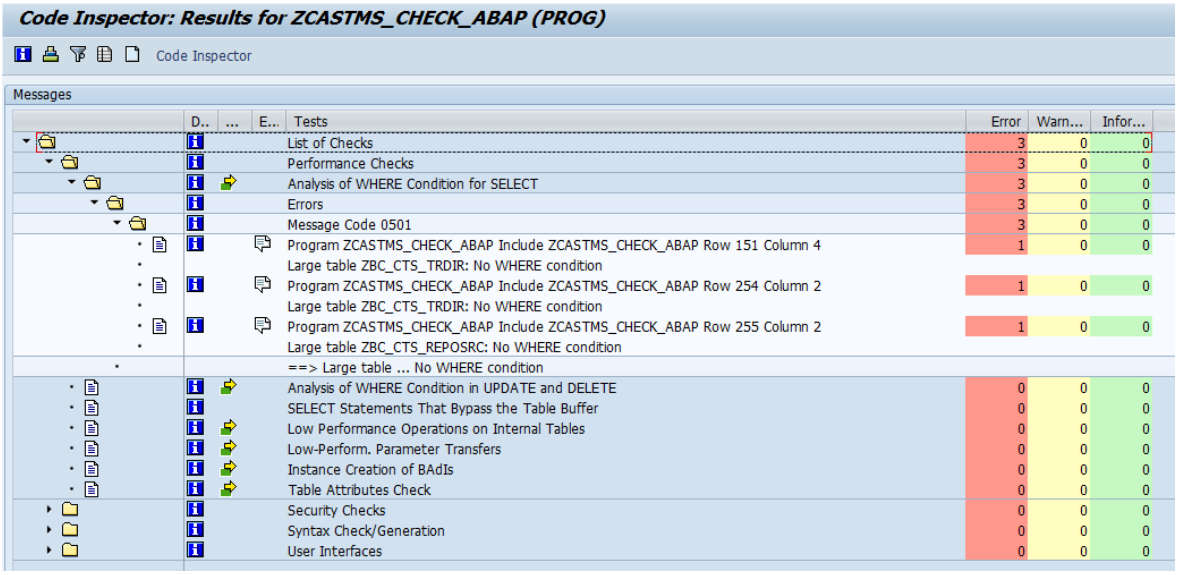
BUCHALCEVOVÁ, A.; Metodiky vývoje a údržby informačních systémů. 1. Vyd. Praha: Grada Publishing, 2004. 164 s. ISBN 80-247-1075-7

SODOMKA, P.; KLČKOVÁ, H. Informační systémy v podnikové praxi. 1. vyd. Praha: Computer Press, 2011. 504 s. ISBN 978-80-251-2878-7

SCHÄFER, M. SAP Solution Manager. 1. vyd. SAP PRESS, 2009. ISBN 978-1-59229-388-9

Seznam Příloh

Příloha A – Výsledky kontroly Code Inspector



Code Inspector: Results for ZCASTMS_CHECK_ABAP (PROG)

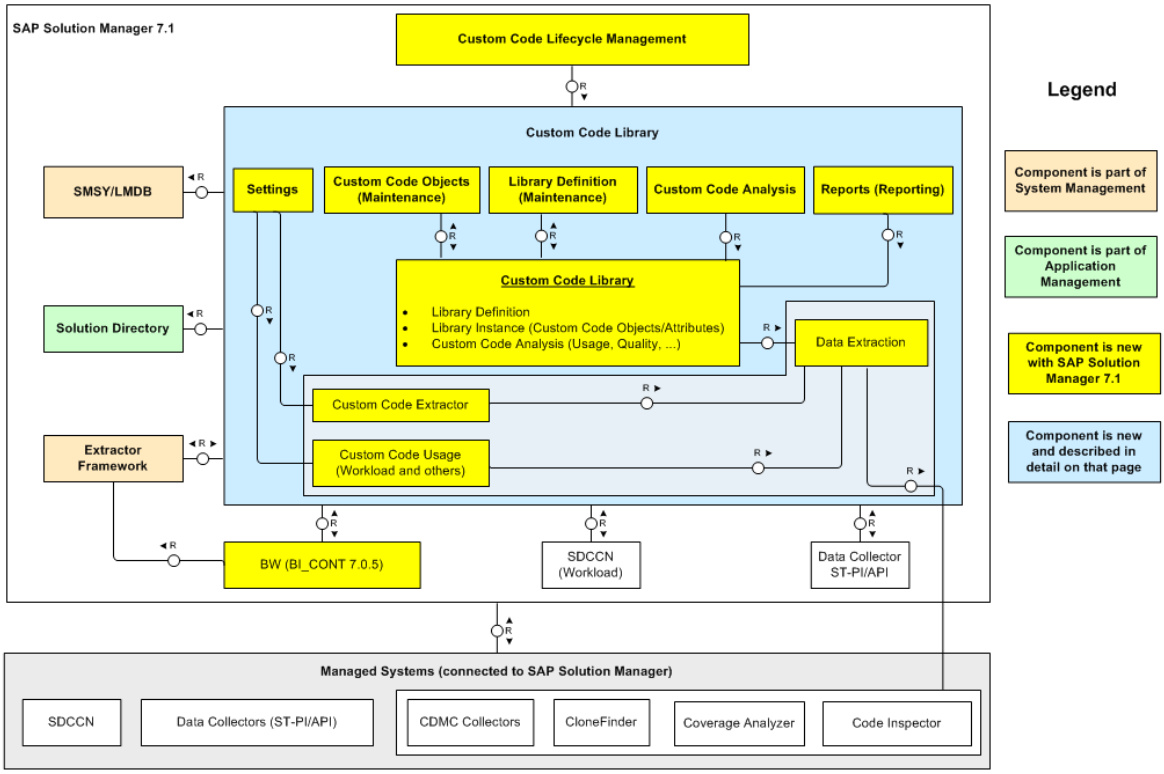
Code Inspector

Messages

D..	...	E...	Tests	Error	Warn...	Infor...
			List of Checks	3	0	0
			Performance Checks	3	0	0
			Analysis of WHERE Condition for SELECT	3	0	0
			Errors	3	0	0
			Message Code 0501	3	0	0
			Program ZCASTMS_CHECK_ABAP Include ZCASTMS_CHECK_ABAP Row 151 Column 4	1	0	0
			Large table ZBC_CTS_TRDIR: No WHERE condition			
			Program ZCASTMS_CHECK_ABAP Include ZCASTMS_CHECK_ABAP Row 254 Column 2	1	0	0
			Large table ZBC_CTS_TRDIR: No WHERE condition			
			Program ZCASTMS_CHECK_ABAP Include ZCASTMS_CHECK_ABAP Row 255 Column 2	1	0	0
			Large table ZBC_CTS_REPOSRC: No WHERE condition			
			==> Large table ... No WHERE condition			
			Analysis of WHERE Condition in UPDATE and DELETE	0	0	0
			SELECT Statements That Bypass the Table Buffer	0	0	0
			Low Performance Operations on Internal Tables	0	0	0
			Low-Perform. Parameter Transfers	0	0	0
			Instance Creation of BAdIs	0	0	0
			Table Attributes Check	0	0	0
			Security Checks	0	0	0
			Syntax Check/Generation	0	0	0
			User Interfaces	0	0	0

Zdroj: SAP Code Inspector - vlastní zpracování

Příloha B – Architektura CCLM uvnitř SAP Solution Manager



Zdroj: SAP Solution Manager 7.1